FPGA-Based Co-processor for Singular Value Array Reconciliation Tomography

Jack Coyne

September 5, 2007

Abstract

This thesis describes a co-processor system that has been designed to accelerate computations associated with Singular Value Array Reconciliation Tomography (SART), a method for locating a wide-band RF source which may be positioned within an indoor environment, where RF propagation characteristics make source localization very challenging. The co-processor system is based on field programmable gate array (FPGA) technology, which offers a low-cost alternative to customized integrated circuits, while still providing the high performance, low power, and small size associated with a custom integrated solution. The system has been developed in VHDL, and implemented on a Virtex-4 SX55 FPGA development platform. The system is easy to use, and may be accessed through a C program or MATLAB script. Compared to a Pentium 4 CPU running at 3 GHz, use of the co-processor system provides a speed-up of about 6 times for the current signal matrix size of 128-by-16. Greater speed-ups may be obtained by using multiple devices in parallel. The system is capable of computing the SART metric to an accuracy of about -145 dB with respect to its true value. This level of accuracy, which is shown to be better than that obtained using single precision floating point arithmetic, allows even relatively weak signals to make a meaningful contribution to the final SART solution.

Acknowledgements

I would like to thank my advisor, Professor Jim Duckworth for giving me the opportunity to partake in research that is as useful, interesting, and groundbreaking as the Precision Personnel Locator project. I would like to thank all the PPL team members for making this research effort possible. Each member plays a vital role, and I am confident that the fruits of their labor will help to save the lives of many. I would like to pay my respects to emergency responders who have given their lives in the line of duty, and to those who continue to risk their own lives despite these tragedies, and despite the lack of adequate tracking technology. I would like to thank the Department of Justice for funding this project, and helping to provide the best possible safety measures for those who protect the public. Most importantly, I would like to thank my family for providing love, support, and encouragement throughout my life, and for being so understanding during the times when I have neglected all things other than this work. I dedicate this work to my grandfather, John William Wolf, who seeded my interest in engineering.

Contents

List of Tables

List of Figures

Table 1: List of acronyms used in this document

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
BRAM	Block RAM
CLB	Configurable Logic Block
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FIFO	First-In First-Out memory structure
FPGA	Field Programmable Gate Array
LSB	Least Significant Bit
MAC	Media Access Controller
PC	Personal Computer
PCI	The Peripheral Component Interconnect
PPL	The WPI Precision Personnel Locator project
RAM	Random Access Memory
RF	Radio Frequency
RMS	Root Mean Square
SART	Singular Value Array Reconciliation Tomography
SDRAM	Synchronous Dynamic RAM
SRAM	Synchronous RAM
SVD	Singular Value Decomposition
SX55	A specific FPGA device sold by Xilinx Inc.
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VPU	Vector Processing Unit
WPI	Worcester Polytechnic Institute

Chapter 1

Introduction

The ability to track the locations of personnel deployed within a building is desirable in many situations, especially those wherein individuals may become lost or debilitated and require assistance in egress. In an emergency first-response situation, rescue operations often must be carried out within a very short time frame. This is particularly true during fire fighting missions, when a person in need of assistance will almost certainly be located in an environment without breathable air, where temperatures can exceed 300 degrees Celsius, and where the integrity of the surrounding structure is deteriorating rapidly. These factors can reduce the time available for rescue to just minutes. Tracking must therefore occur in real time, with location updates occurring at least every second, regardless of the size of the structure. Furthermore, location estimates must be very accurate such that, for example, a person under egress assistance may be guided to the correct door when presented with many adjacent avenues, and subject to zero-visibility conditions. Obtaining this level of accuracy has been the goal of many efforts world wide, and is the main objective of the project for which this work serves as a component. The goal of this particular component is to obtain the aforementioned real-time performance despite the computational complexity of the proposed location estimation algorithm.

1.1 Indoor Tracking

Tracking of personnel located within a building using a system that requires no preexisting infrastructure is a goal that has been addressed by research teams from academic, government, and commercial institutions all over the world [1] [2] [3] [4] [5]. Despite these efforts and the millions of dollars invested, no system with the required accuracy, reliability, and affordability has been demonstrated, let alone brought to market. Many tracking techniques have been investigated, but the two most promising are radio-based tracking and inertial navigation.

Radio based systems attempt to extract time-of-flight information from radio signals that have propagated through a building, from the object being tracked to externally located reference units, or visa versa. The major challenge posed by this technique results from the fact that radio-frequency electromagnetic waves are reflected and diffracted by metallic structural members and metallic objects within the building. These phenomena result in a received signal that can be defined as the sum of the direct-path component and the reflected, "multi-path", components. Solving for the range to the object being tracked requires isolating the direct-path component. Separating this component from the undesired portion of the received signal is not a trivial task and presents the most significant challenge to research teams developing radio-based location systems [5].

Inertial navigation systems rely on sensors capable of measuring linear and rotational acceleration in order to determine velocity and position. The adoption of this technique is inhibited mainly by the cost of a functional system, as well as regulations that control the availability of high-precision sensors. Affordable inertial sensors do not have the accuracy required for long-duration tracking. Small amounts of bias and drift in the outputs of these sensors lead to the accumulation of large errors as these outputs are integrated over time to obtain the final position estimate [4]. Highly accurate sensors are not only larger and more expensive, but they are also subject to export restrictions imposed by the government due to their suitability to weapons applications, such as missile guidance systems [6].

1.2 The Precision Personnel Locator System

The Worcester Polytechnic Institute Precision Personnel Location (PPL) project began in December, 1999, prompted by the tragic loss of six firefighters in the Worcester Cold Storage Warehouse fire. The deaths of these men have been attributed in part to the inability to determine their locations within the burning structure [7]. The goal of the PPL project is to develop a system capable of providing sub-meter accuracy in the location and tracking of personnel situated within hazardous indoor environments. The system is primarily radio-based due to the limitations of available inertial sensors, and requires no pre-existing infrastructure so that it may be used in any location [8].

Over a dozen faculty and student team members, participating in seven years of research efforts, have produced four generations of prototype hardware, and many more generations and refinements of the signal processing algorithms developed for estimating location using the received radio signals. The performance of the system has matured from coarse outdoor tracking using 15 MHz of bandwidth [9], to a 150 MHz system that enables indoor tracking with errors on the order of one meter [8]. In early 2006 it was apparent that the final project goal was within reach, and the time came to consider the deployment of the system, and how this would be made feasible.

One of the most evident obstacles on the path towards deployment was the burden imposed by the computational complexity of the position estimation algorithm, known as Singular Value Array Reconciliation Tomography (SART). The amount of computation associated with the SART algorithm is so great that scanning a 20m by 20m room for a single target takes approximately 1 second using a Pentium 4 CPU running at 3GHz. Using this figure, and assuming that a structure consists of many rooms and many floors, and that there could potentially be dozens of personnel that need to be located or tracked, a complete scan could easily take many minutes.

Acceleration of the SART algorithm could be accomplished using a cluster of PCs working in parallel, but this solution would be physically too large for a mobile application, and needlessly power hungry. A smaller more efficient solution would be one that uses custom hardware instead of the generalized architecture of a personal computer. For customizable high-speed signal processing, Field Programmable Gate Arrays (FPGAs) are the industry

standard hardware. They are second in performance only to Application Specific Integrated Circuits (ASIC), which can easily cost millions dollars to design and test, yet FPGAs cost only a few thousand dollars [10]. An FPGA-based system was chosen as the platform for the SART accelerator system described here.

1.3 Outline

This thesis begins with the presentation of the SART technique for indoor tracking in Chapter 2. The algorithm is broken into stages so that it may be easily partitioned into tasks that are appropriate for hardware and software implementation. The number of computations involved in each stage are quantified to motivate the need for hardware-based acceleration, and provide a basis for intelligent partitioning decisions. The FPGA development platform, which was used for prototyping, is described in Chapter 3. An overview of FPGA technology and the architectural details of the selected FPGA are presented, along with some important architecture related design considerations. The implementation of the SART co-processor system is presented in Chapter 4. Chapter 5 contains a description of how the SART co-processor system is operated. In Chapter 6, the performance of the SART co-processor system is analyzed in terms of accuracy, and speedup. The final chapter presents conclusions, and avenues for future work.

Chapter 2

Singular Value Array Reconciliation Tomography

This chapter contains a description of a new technique for radio-based indoor location. The technique was developed at WPI for the purpose of tracking first-response personnel such as firefighters. An overview of the technique is presented first. Later sections provide a more detailed description of the algorithm, and quantify the number of computations associated with each processing stage.

2.1 Overview

Singular value Array Reconciliation Tomography (SART) [11] is a source localization technique developed as part of the PPL project. The SART algorithm was developed as a means for locating the origin of a wide-band multi-tone signal using a receiver array with arbitrary recieve-element geometry. The performance of the SART technique is described only briefly here, because ultimately, this thesis is not concerned with why or how well SART works, but only with accomplishing it quickly. A more through explanation and performance analysis of SART can be found in documents that deal specifically with algorithm development [12].

The location process begins with the transmission of a multi-tone signal by the device that is to be located. The transmission is recorded at multiple receiving elements, which ideally surround the transmitter. For the PPL project application of SART, these receiving elements correspond to antennas located on fire-trucks and other vehicles located around the outside of a building in which personnel are being tracked. The received signals are digitized using an analog to digital converter and stored in memory so that they may be used as inputs to the SART algorithm. The remaining inputs to the algorithm are the relative coordinates of the receive elements, which are assumed to be known.

The SART algorithm is an imaging technique (hence the use of the term tomography). Given the algorithm inputs at any given instance, the physical area of interest is scanned at regular spatial intervals, where it is assessed in terms of the SART metric. This set of scan locations is known as the scan-grid. The location of maximum metric value indicates the estimated location of the transmitter. Scanning is accomplished by applying negative time delays to the received signals. These delays corresponding to the light-speed travel time between the current scan location and each receive element. Because the locations of the receive elements are known, these time delays can be easily calculated for each location on the scan-grid. Time delay application is accomplished by imposing a linear (with frequency) phase shift to the signals from each receive element.

If the exact location of the transmitter is scanned, then the application of the negative time delays will counter-act the actual time delays associated with the propagation of the multi-tone signal from the transmitter to the receive elements. The result of this will be temporal alignment of all of the received signals, thereby maximizing their linear dependence in the ideal, zero multi-path case. For this reason, the imaging metric chosen for SART is the level of linear dependence between the rephased signals. The level of linear dependence is measured by assigning each rephased input signal to a different column in a matrix, performing singular value decomposition of that matrix, and observing the first singular value[12].

Using the process of Singular Value Decomposition (SVD), any matrix can be broken down into a set of unitary column and row vectors, and a corresponding set of scalar values. These components contain all of the information necessary to reconstruct the original matrix. The singular vectors serve as an orthonormal basis for the matrix, while the singular values contain information about the amplitudes of these vectors in their role as components within the original matrix. Together, the singular values provide an indication of matrix rank [13].

For example, if a matrix is composed of multiple linearly dependent columns (or rows), then the matrix is said to have a rank of one (i.e. all but the first singular value will be zero). If there are small dissimilarities between columns then the SVD will reveal more non-zero singular values, with magnitudes corresponding to the amplitudes of the components that impart these dissimilarities. Larger dissimilarities will be reflected in larger rank supporting singular values.

If dissimilarities arise due to the application of linear phase, as they do in the SART scanning procedure, then the increase in the lesser singular values occurs with a corresponding decrease in the first (largest) singular value. This is because the Frobenius norm of a matrix (which can be calculated as the square-root of the sum of the squares of all the singular values) is maintained during such an operation [12]. Leveraging this property of singular values, the SART algorithm uses singular value decomposition to obtain a measure of linear dependence between columns of the signal matrix in the form of the first singular value.

In SART, the assumption is that each receive element will see the direct-path component of the transmitted signal and various multi-path components. The hope is that different receive elements will collect multi-path components that are of lower linearly dependence and/or amplitude than the collected direct-path components, when analyzed according to the scanning procedure. If this assumption holds, then the SART metric will be maximized in the region of the transmitter. Many rounds of simulation and live testing have been conducted, producing good results that have verified the validity of the SART method [12]. The major disadvantage of the SART algorithm is its computational complexity, most of which arises due to the application of the SVD.

The following sections provide a detailed description of the SART algorithm in an attempt to reveal the large computational burden it presents, and partition it into a manageable set of tasks. This will serve to motivate and outline the work described in this document, which represents an effort to accelerate this algorithm using custom digital signal processing hardware. The description will include an enumeration of the singular value decomposition process because it represents a substantial portion of SARTs computational complexity, and because (as will be revealed in later sections) the partitioning of the algorithm between the accelerator and the host PC was chosen such that the SVD is computed partially in each partition.

2.2 Signal Propagation

As mentioned in the SART overview section, any person being tracked using the PPL system must wear a radio transmitter that broadcasts a multi-tone signal. The signal is generated in baseband by playing a predefined waveform through a digital-to-analog converter (DAC), up-converted to radio frequency (RF), and then filtered to obtain a single side-band transmission of the form

$$s(t) = \sum_{i=1}^{m} e^{j(2\pi f_i t + \theta_i)} \tag{2.1}$$

where f_i and θ_i represent the frequency and initial phase of the i^{th} RF tone (or subcarrier).

The RF signal propagates from the transmitter, through the building, to each of the receivers. The travel-time associated with propagation can be calculated from the distance of travel and the speed of light in the propagation medium. Assuming the propagation occurs primarily in air, the propagation velocity may be approximated with the propagation velocity in a vacuum, 299,792,458 meters per second. If multi-path components are ignored, then the received signal is identical to the transmitted signal, but delayed by the travel-time. The signal received at a distance, d, from the transmitter can be represented as

$$s(t) = \sum_{i=1}^{m} e^{j(2\pi f_i (t - \frac{d}{c}) + \theta_i)} \tag{2.2}$$

where c is the speed of propagation.

At each receive element the received signal is down-converted to baseband, digitized using an analog-to-digital converter (ADC), and stored for processing. This is where the application of the SART algorithm begins.

2.3 The SART Signal Matrix

The first step in the SART algorithm is to obtain the frequency-domain representation of the received signal. This allows the sub-carrier tones to be isolated from each other, and from noise components at other frequencies. The conversion is accomplished using the Discrete Fourier Transform (DFT), implemented using the Fast Fourier Transform (FFT) algorithm. The sub-carrier tones are placed at even intervals, which coincide with the frequency bins of the transform. This prevents leakage of the sub-carrier energy into adjacent frequency bins [12]. The output of the FFT is an array of complex values that represent the amplitudes and phases of the frequency components that comprise the input time-domain signal. The values that correspond to the transmitted sub-carrier frequencies are saved. Values from other frequency bins represent noise and interfering signals and are therefore discarded.

The transmitted signal can be represented as an array of complex values that encode the magnitude and phase each of the sub-carrier tones. Each received signal can be represented as this same signal with the application of a time delay associated with propagation. In the frequency domain, this time delay appears in the form of a phase shift that decreases linearly with frequency. If the transmitted signal is represented in the form

$$S = \begin{bmatrix} s_1 & s_2 & s_3 & \dots & s_m \end{bmatrix} \tag{2.3}$$

where m equals the number of sub-carrier tones, and s_i encodes the arbitrary phase of the transmitted signal for the i^{th} sub-carrier, then the received signal at a distance, d, from the transmitter can be represented as

$$S = \begin{bmatrix} s_1 e^{-j\mathbf{2}\pi\Delta f\frac{d}{c}} & s_2 e^{-j\mathbf{4}\pi\Delta f\frac{d}{c}} & s_3 e^{-j\mathbf{6}\pi\Delta f\frac{d}{c}} & \dots & s_m e^{-j2\mathbf{m}\pi\Delta f\frac{d}{c}} \end{bmatrix} \tag{2.4}$$

where Δf represents the frequency spacing between sub-carrier tones (and for convenience the frequency of the first tone), and the exponential factors encode the frequency-dependent phase shift undergone by each of the m sub-carriers due to propagation delay. Since there are many receivers, each at a different distance from the transmitter, the group of received signals can be formed into a matrix, with each column representing the signal recorded at

one receive element. The SART signal matrix has the form

$$\mathbf{S} = \begin{bmatrix} s_1 e^{-j2\pi\Delta f \frac{d_1}{c}} & s_1 e^{-j2\pi\Delta f \frac{d_2}{c}} & \cdots & s_1 e^{-j2\pi\Delta f \frac{d_n}{c}} \\ s_2 e^{-j4\pi\Delta f \frac{d_1}{c}} & s_2 e^{-j4\pi\Delta f \frac{d_2}{c}} & \cdots & s_2 e^{-j4\pi\Delta f \frac{d_n}{c}} \\ s_3 e^{-j6\pi\Delta f \frac{d_1}{c}} & s_3 e^{-j6\pi\Delta f \frac{d_2}{c}} & \cdots & s_3 e^{-j6\pi\Delta f \frac{d_n}{c}} \\ \vdots & \vdots & \ddots & \vdots \\ s_m e^{-j2m\pi\Delta f \frac{d_1}{c}} & s_m e^{-j2m\pi\Delta f \frac{d_2}{c}} & \cdots & s_m e^{-j2m\pi\Delta f \frac{d_n}{c}} \end{bmatrix} \tag{2.5}$$

where d_k represents the distance from the transmitter to the k^{th} receiver. It is this signal matrix that is rephrased according to the scan-grid and evaluated in terms of the SART metric.

2.4 SART Scan Rephasing

After the signal matrix has been constructed, the SART scanning procedure begins. It is this exhaustive imaging approach that leads to the immense number of computations required to produce a single location estimate. The spatial resolution, or density, of the scan-grid is chosen based on bandwidth [12], and is typically 0.25 to 0.5 meters for the current 150 MHz implementation. Therefore, even a modest 20m-by-20m-by-20m volume will contain 8000 scan locations. As a more realistic example, a standard street block in the Manhattan has a footprint of about 80m-by-270m, and assuming a modest height of about 25m, this volume would contain over 500,000 scan locations. It is true that the number of locations being scanned can be reduced if the previous location of a target is known, and only the surrounding region is searched. However, the number of scan locations will remain high when tracking dozens of independent targets, and will grow larger when using the higher bandwidths needed for better accuracy.

Scanning is accomplished by applying a linear phase shift to each column of the signal matrix. The phase slope is calculated according to the propagation delay between the receiver corresponding to that column and the current scan location. This delay is easily obtained, if the geometry of the receivers is known, by calculating the Euclidean distance between the receiver and the scan location and then dividing by the speed of propagation. Using the

received signal matrix form from (2.5), the rephrased signal matrix has the form:

$$\mathbf{S} = \begin{bmatrix} s_1 e^{-j2\pi\Delta f \frac{d_1}{c}} e^{j2\pi\Delta f \frac{d_{s1}}{c}} & s_1 e^{-j2\pi\Delta f \frac{d_2}{c}} e^{j2\pi\Delta f \frac{d_{s2}}{c}} & \cdots & s_1 e^{-j2\pi\Delta f \frac{d_n}{c}} e^{j2\pi\Delta f \frac{d_{sn}}{c}} \\ s_2 e^{-j4\pi\Delta f \frac{d_1}{c}} e^{j4\pi\Delta f \frac{d_{s1}}{c}} & s_2 e^{-j4\pi\Delta f \frac{d_2}{c}} e^{j4\pi\Delta f \frac{d_{s2}}{c}} & \cdots & s_2 e^{-j4\pi\Delta f \frac{d_n}{c}} e^{j4\pi\Delta f \frac{d_{sn}}{c}} \\ s_3 e^{-j6\pi\Delta f \frac{d_1}{c}} e^{j6\pi\Delta f \frac{d_{s1}}{c}} & s_3 e^{-j6\pi\Delta f \frac{d_2}{c}} e^{j6\pi\Delta f \frac{d_{s2}}{c}} & \cdots & s_3 e^{-j6\pi\Delta f \frac{d_n}{c}} e^{j6\pi\Delta f \frac{d_{sn}}{c}} \\ \vdots & \vdots & \ddots & \vdots \\ s_m e^{-j2m\pi\Delta f \frac{d_1}{c}} e^{j2m\pi\Delta f \frac{d_{s1}}{c}} & s_m e^{-j2m\pi\Delta f \frac{d_2}{c}} e^{j2m\pi\Delta f \frac{d_{s2}}{c}} & \cdots & s_m e^{-j2m\pi\Delta f \frac{d_n}{c}} e^{j2m\pi\Delta f \frac{d_{sn}}{c}} \end{bmatrix} \quad (2.6)$$

where d_{sk} is the distance between receiver k and the scan location.

If the distance to the scan location equals the true distance between a receiver and the transmitter (i.e. $d_{sk} = d_k$), then the two exponential factors will cancel for that receiver. Furthermore, if the true location of the transmitter is scanned, then cancellation will happen for all receivers. In this case, the columns of the rephrased signal matrix will become identical, apart from any multi-path or noise components. Ideally, this will lead to a maximization of the first singular value at the scan location corresponding to the true location of the transmitter.

2.5 Singular Value Decomposition

Through the process of singular value decomposition, a matrix, $\mathbf{A}$, is broken-down into two orthonormal matrices, $\mathbf{U}$ and $\mathbf{V}$, and a diagonal matrix $\mathbf{\Sigma}$, such that

$$\mathbf{U\Sigma V}^H = \mathbf{A}$$

where $\mathbf{V}^H$ indicates the complex conjugate transpose of $\mathbf{V}$.

The columns of $\mathbf{U}$ and $\mathbf{V}$ contain the left and right singular vectors for $\mathbf{A}$, which form an orthonormal basis for $\mathbf{A}$. The diagonal of $\mathbf{\Sigma}$ contains the singular values of $\mathbf{A}$, which hold magnitude or scaling information. In this way, the SVD allows $\mathbf{A}$ to be re-expressed as a diagonal matrix through a change of basis, revealing important information about the composition of $\mathbf{A}$ [13]. In the SART application, the singular values reveal information about the rank of the signal matrix.

The SART imaging metric is the first singular value of the rephased signal matrix. Therefore, singular value decomposition must be performed once for each location on the scan-grid. The complexity of the SVD algorithm for an m-by-n matrix is $O(4nm^2 + 22n^3)$ operations [13], so the computational burden presented by this workload is quite large, and grows rapidly with the size of the matrix. Fortunately, because the SART algorithm requires only the singular values, and not the singular vectors, this work load can be substantially decreased to $O(2mn^2 + 2n^3)$ operations [13]. The process of computing the singular values of a matrix is described below.

The singular values of a matrix may be obtained through the process of diagonalization, where all but the diagonal elements of the matrix are reduced to zero (or annihilated). This is typically accomplished through unitary transformations, which preserve matrix rank and the energy possessed by each linearly independent component. Most frequently, these are either Householder transformations, or Givens rotations [13]. The diagonalization process can be divided into three stages [13]:

- QR decomposition, wherein the matrix, $\mathbf{A}$, is factored into an orthogonal matrix, $\mathbf{Q}$, and an upper triangular matrix, $\mathbf{R}$, such that $\mathbf{QR} = \mathbf{A}$. The matrix $\mathbf{R}$ retains the same singular values as $\mathbf{A}$ while having fewer total non-zero elements.

- Bidiagonalization of the matrix $\mathbf{R}$, such that the resulting matrix, $\mathbf{B}$, contains non-zero values in only its diagonal elements, and the elements just above the diagonal. The singular values of $\mathbf{A}$ are still retained.

- Diagonalization of $\mathbf{B}$ is the final step of reducing the input to its diagonal form, $\mathbf{\Sigma}$, in which the singular values of the original input matrix lie along the diagonal.

This is a convenient division of the process for two reasons.

- For rectangular matrices, especially where $m \gg n$ or $n \ll m$, the QR decomposition stage results in a substantial reduction in the total size of the input matrix, particularly when only singular values are desired and $\mathbf{Q}$ can be discarded. This is advantageous because it reduces the amount of computer memory needed to store the matrix elements, and eliminates needless computations during bidiagonalization.

- QR decomposition and bidiagonalization both require a deterministic number of op-

erations, which can be calculated based on the size of the input matrix. Diagonalization, on the other hand, is an iterative process which terminates based on convergence conditions. Therefore diagonalization has a non-deterministic run-time, which is influenced by the content of the matrix. Separating the deterministic stages from the non-deterministic stages is an important design consideration when partitioning the SVD process between custom parallel hardware and the sequential host CPU.

2.5.1 QR Decomposition

QR decomposition is the process of factoring a matrix, $\mathbf{A}$, into an orthogonal matrix, $\mathbf{Q}$, and an upper-triangular matrix, $\mathbf{R}$, such that $\mathbf{QR} = \mathbf{A}$. When performing the decomposition in order to compute the singular values of $\mathbf{A}$, only $\mathbf{R}$ is needed [13]. In this case it may be more appropriate to call the process triangularization, or unitary triangularization [14]. The term QR decomposition is maintained here because its use appears to be more widespread. Two common methods for obtaining the QR decomposition are by use of Householder reflections and by use of Givens rotations.

2.5.2 Householder Method

Using a Householder transformation [14], a vector may be reflected about an arbitrary plane (in the 3-dimentional case), or hyperplane in the N-dimensional case. If this hyperplane is chosen correctly, then the resulting vector can be made to lie directly along one axis of the coordinate system, or have components along some dimensions but not others. This kind of transformation can also be applied to a matrix, where each column of the matrix corresponds to a different vector. In this case all vectors are reflected about the same hyperplane, and because the orientations of the vectors relative to each other are maintained, the singular values of the matrix will be preserved.

In order to perform QR decomposition, a group of vectors comprising a matrix may be successively reflected about a sequence of hyperplanes. The first hyperplane may be chosen such that the reflection operation causes the vector corresponding to the first column of the matrix to be aligned along the dimension corresponding to the first row of the matrix, thereby

causing all elements in the first column that lie below the first row to be annihilated. Likewise, the second Householder transformation may be chosen in a way that causes the vector in the second column to be reflected such that it has components in only two dimensions, those corresponding to the first two rows of the matrix. This process may continue until all sub-diagonal elements in the matrix are zero, and the matrix is upper-triangular [14].

In terms of implementation, the Householder method is typically considered the most efficient for implementation on a sequential processor, because it involves fewer total operations than methods using Givens rotations [15][16]. Unfortunately the algorithm involves data dependencies and other characteristics that make the method difficult to implement using parallel processing techniques [13]. Consideration of the Householder method for use in the SART co-processor system revealed two such problems that would have hindered an FPGA-based QR decomposition implementation employing this method.

The first problem lies in the fact that one reflection operation must be performed before the Householder transformation matrix for next reflection operation can be computed, thereby forcing a linear sequence of: compute Householder matrix, perform transformation, compute Householder matrix, perform transformation, et cetera. In some cases, a dependency such as this can be overcome through pipelining techniques, whereby the first transformation is applied to the input matrix while the second transformation is being applied to the previous input matrix, and the third transformation is being applied to the matrix before that, et cetera. Unfortunately there are problems with this approach towards the Householder method on an FPGA.

The main concern relates to the amount of memory available to each stage in the pipeline. That is, each stage must store the entire contents of the matrix, apart from the elements annihilated by previous stages. This may not be a problem for multiprocessor computer systems, but even in modern FPGAs, memory resources can be quite limited. The use of off-chip memory may be a solution, but this would lead to a requirement for very large IO bandwidth, which is impractical. The second flaw in the pipeline solution is a lack of balance between the loads on each stage. The nature of the Householder method is such that each stage would be required to work on a smaller matrix than the previous stage, meaning that some stages would finish their computations more quickly than other stages, and would be left idle for some time, wasting hardware resources.

Another characteristic of the Householder method that makes its implementation less practical is its requirement for the computation of square-roots and division operations when calculating vector norms, and unit vectors. While not impossible to implement on an FPGA, these operations typically involve more hardware and latency than multiplication and addition operations, and should be avoided. For the reasons outlined above, QR decomposition methods involving Givens rotations are typically preferred for parallel implementations, despite the fact that the Householder method requires fewer total operations.

2.5.3 Givens Rotations

During a Givens rotation, an N-dimensional vector is rotated such that its components in only two dimensions are changed. Similar to a Householder transformation, the Givens rotation angle may be chosen such that the vector being rotated is left in an orientation where one of its components is zero, while its overall length remains unchanged. If the vector is part of a matrix, and all vectors in the matrix are rotated by the same angle, then one element of the matrix may be reduced to zero without effecting the singular values of the matrix. The Givens method for QR decomposition of a matrix involves a sequence of rotations, with each rotation eliminating one element of the matrix [15].

A Givens rotation is performed by premultiplication with a rotation matrix of the form

$$G = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{2.7}$$

Because the rotation matrix is sparse, the entire matrix multiplication need not be implemented. Instead, only the rows or columns involved in the operation are altered. For

example,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a'_{21} & a'_{22} & a'_{23} & a'_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a'_{41} & a'_{42} & a'_{43} & a'_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 0 & 1 & 0 \\ 0 & \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{2.8}$$

becomes

$$\left.\begin{array}{l} \begin{bmatrix} a'_{21} & a'_{22} & a'_{23} & a'_{24} \end{bmatrix} = \cos(\theta) \begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} - \sin(\theta) \begin{bmatrix} a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \\ \qquad\qquad and \\ \begin{bmatrix} a'_{41} & a'_{42} & a'_{43} & a'_{44} \end{bmatrix} = \cos(\theta) \begin{bmatrix} a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} + \sin(\theta) \begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \end{array}\right\} \tag{2.9}$$

For the Givens rotation to eliminate a_{42}, θ must be chosen such that

$$cos(\theta)a_{42} = -sin(\theta)a_{22} \tag{2.10}$$

Therefore,

$$\theta = tan^{-1}(\frac{-a_{42}}{a_{22}}) \tag{2.11}$$

A Givens rotation may also be applied by post-multiplication with a rotation matrix. In this case, the operation involves two columns of **A** instead of two rows.

In the case of a matrix containing complex values, the phase of one or both rows must be adjusted such that element annihilation still occurs. For example, the previous result

becomes

$$\left.\begin{array}{c} \left[\begin{array}{cccc} a'_{21} & a'_{22} & a'_{23} & a'_{24} \end{array}\right] = \cos(\theta)e^{-j\phi_{22}} \left[\begin{array}{cccc} a_{21} & a_{22} & a_{23} & a_{24} \end{array}\right] - \sin(\theta)e^{-j\phi_{42}} \left[\begin{array}{cccc} a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \\ \\ and \\ \\ \left[\begin{array}{cccc} a'_{41} & a'_{42} & a'_{43} & a'_{44} \end{array}\right] = \cos(\theta)e^{-j\phi_{42}} \left[\begin{array}{cccc} a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] + \sin(\theta)e^{-j\phi_{22}} \left[\begin{array}{cccc} a_{21} & a_{22} & a_{23} & a_{24} \end{array}\right] \end{array}\right\} \quad (2.12)$$

where ϕ_{22} is the phase angle of a_{22} and ϕ_{42} is the phase angle of a_{42}.

Order is critical when implementing the sequence of rotations. Rotations must be performed in a manner that avoids reintroducing energy into a matrix element that was reduced to zero by a previous rotation. This can be accomplished by performing the rotations starting from the left most column of the input matrix, and moving to the right only after all sub-diagonal elements in the current column have been eliminated. For example if the input matrix is

$$\mathbf{A} = \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{array}\right] \quad (2.13)$$

and a rotation is performed to move energy from element a_{31} to element a_{11}, then the top and bottom rows must be altered, and the result is

$$\mathbf{A} = \left[\begin{array}{ccc} a'_{11} & a'_{12} & a'_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & a'_{32} & a'_{33} \end{array}\right] \quad (2.14)$$

If the next rotation attempts to move energy from element a'_{32} to element a'_{12}, then the bottom row will again be altered. In the process, energy from a'_{11} may be transferred back to the lower left element. Instead, energy from a_{21} should be transferred to element a'_{11}, in order to obtain

$$\mathbf{A} = \left[\begin{array}{ccc} a''_{11} & a''_{12} & a''_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{array}\right] \quad (2.15)$$

Now, energy from a'_{32} may be moved to element a'_{22}. Because the two sub-diagonal elements in the left column are both zero, there is no danger of reintroducing energy into either of them. After this rotation, the example 3-by-3 matrix will be triangular. This process may be extended to perform triangularization of larger matrices. The total number of rotations required is equal to the number of sub-diagonal elements in the matrix.

The most attractive aspect of the Givens rotation method of QR decomposition is its suitability for parallel implementation. Because only two rows or two columns of the matrix are altered during any one rotation, multiple rotations may occur in parallel provided they do not involve any of the same matrix elements. Furthermore, square-root and division operations may be avoided in a hardware implementation by employing the COordinate Roatation DIgital Computer (CORDIC) algorithm for calculating 2-D vector magnitudes, and unit-length rotation vectors.

2.5.4 The CORDIC Algorithm

The COordinate Rotation DIgital Computer (CORDIC) algorithm is widely used for digital signal processing systems implemented in hardware [17]. The algorithm allows trigonometric functions to be computed without square-root or division operations. Instead, the algorithm employs a series of shift and add operations to iteratively rotate an input vector to zero, or through some arbitrary angle [18].

Consider a two-dimensional rotation of the form

$$\begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{2.16}$$

Division by $cos(\theta)$ gives

$$\frac{1}{cos(\theta)} \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} 1 & -tan(\theta) \\ tan(\theta) & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{2.17}$$

If the rotation angle is chosen such that $tan(\theta) = 2^{-n}$, then multiplication by $tan(\theta)$ can be

implemented as a binary shift, and a vector rotation becomes

$$\left.\begin{array}{c} \frac{1}{cos(atan(2^{-n}))}a_1' = a_1 \mp a_2(2^{-n}) \\ \\ and \\ \\ \frac{1}{cos(atan(2^{-n}))}a_2' = a_2 \pm a_1(2^{-n}) \end{array}\right\} \tag{2.18}$$

where the $\pm$ operations are used to encode the direction of rotation.

Using this method for rotation, an input vector may be rotated by successively smaller angles, until the desired angle is reached. Rotation towards zero may be achieve by observing the sign of x and y, and choosing the rotation direction accordingly. Correction for the $\frac{1}{cos}$ scale factor may be neglected until the end of the series of rotations, at which time the compounded correction factor may be applied as multiplication by a pre-computed constant

$$K(n) = \prod_{i=0}^{n} cos(atan(2^{-i})) \tag{2.19}$$

where n is the number of so called mirco-rotation iterations. Typically, one micro-rotation is needed for each bit of angular precision.

2.5.5 Bidiagonalization

After QR decomposition has been performed in order to obtain an upper-triangular matrix, bidiagonalization may begin. The purpose of bidiagonalization is to bring the matrix as close its diagonal form as possible while still employing an algorithm with a deterministic run-time. Another benefit of bidiagonalization is the elimination of phase information. That is to say, a complex matrix may be bidiagonalized such that all non-zero elements are strictly real. This simplifies the subsequent diagonalization process, and reduces the amount of data transferred between the two processing stages.

The bidiagonalization is very similar to the QR decompositon process in that it can be performed using a sequence of Givens rotations to reduce all but the diagonal and superdiagonal

elements to zero. As in the QR decomposition case, reintroduction of energy into previously annihilated elements must be avoided. This may be accomplished by alternating between column elimination and row elimination.

The process of bidiagonalizing a upper-triangular matrix begins with the elimination of the upper-rightmost matrix element using a Givens rotation from the right. This corresponds to an operation involving two matrix columns. The energy from the upper-right may be rotated into any column other than the left-most column, which contains zeros below the diagonal (due to the QR decomposition stage) that should not be corrupted. Unfortunately, rotation operations that seek to eliminate elements in the first row will re-introduce energy into the sub-diagonal elements of other columns of the matrix, but this counter-productivity is unfortunately necessary. Rotations continue in this fashion until all elements in the top row, apart from the two leftmost elements, are annihilated. At this point the matrix should have the form

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \tag{2.20}$$

The next phase is the elimination of the sub-diagonal elements in the second column. When these elements have been annihilated, the second row may be targeted. All elements in the second row apart from the diagonal element and the element just to the right of the diagonal element may be eliminated without introducing energy into the previously annihilated elements. The algorithm proceeds with the elimination of sub-diagonal elements in column three and elements to the right of the super-diagonal in row four, and continues until the matrix is bidiagonal.

2.5.6 Diagonalization

Diagonalizing the bidiagonal matrix is the final step in determining the singular values of the original input matrix. Many methods may be employed, but the simplest one involves the same Givens rotations as the QR decomposition and bidiagonalization algorithms. More efficient methods have been developed in [19], [20], and others. Regardless of which method is used, diagonalization is an iterative process. Each time an off-diagonal is annihilated, a non-zero value is reintroduced into a previously annihilated element. Fortunately, the process converges after several iterations [13].

2.6 Quantified SART Computational Requirements

The computational burden associated with the SART algorithm may be described as the number of arithmetic operations required for a single SART scan. This figure can be itemized into a list of operation counts for each stage. Another important metric is the amount of data transferred between stages, which has implications related to memory and interface bandwidth requirements. These operations counts and bandwidth requirements, which are discussed below, will be functions of the size of the signal matrix, m-by-n, where n is the number of receive elements, and m is the number of sub-carrier tones in the transmitted signal. Some of them will also be functions of the number of locations in the SART scan-grid, G, or the number of ADC samples, N, collected at each receive element.

Fast Fourier Transform (FFT): The FFT algorithm for performing a Discrete Fourier Transform (DFT) has a complexity of $O(Nlog_2(N))$ [21], where N is the length of the input array. For the SART application this length is equal to the number of time samples collected at each receive element. Because an FFT must be performed for each receiver, a factor of n must be included, resulting in a total complexity of $O(nNlog_2(N))$. The amount of data that flows into this stage of the algorithm is nN. Because only the frequency bins corresponding to sub-carrier tones are used, the amount of data flowing out of this stage equals the number of elements in the signal matrix, mn.

Signal Matrix Rephasing: The rephasing stage involves element-wise multiplication of the signal matrix with a different rephasing matrix for each scan-grid location. Assuming each complex multiplication requires 6 operations, the complexity of this stage is $O(6Gmn)$. The amount of data in is mn. The amount of data out is Gmn.

QR Decomposition: Using a figure from [13], the complexity of QR decompositon using Givens rotations is $O(12n^2(m - n/3))$. Multiplying by the number of locations in the SART scan-grid gives a total complexity of $O(12Gn^2(m - n/3))$. The amount of data in is Gmn. The amount of data out is less, due to the elimination of the lower portion of the matrix. The amount of data flowing out of this stage is G multiplied by number of non-zero elements in an n-by-n upper triangular matrix, $G(n^2 + n)/2$.

Bidiagonalization: The number of operations required to compute bidiagonalization using Givens rotations is approximately twice the complexity of QR decomposion of an equally sized matrix. The total complexity of the bidiagonalization stage is $O(16Gn^3)$. The amount of data in is $G(n^2 + n)/2$. The amount of data out is equal to one diagonal and one super diagonal for each scan-grid location, or $G(2n - 1)$.

Diagonalization: According to [13], the complexity of diagonalization is $O(54n)$ per iteration, assuming single precision, and approximately 12 operations per square-root (Intel processors generate 2 bits of precision each clock cycle for square root calculations [22]). For a scan-grid with G scan locations, and assuming a conservative 20 interations, the total complexity is $O(1000Gn)$. The amount of data in is $G2n - 1$. The amount of data out is Gn.

Totals

In the current implementation of SART, sixteen receive elements are used, and 103 subcarrier tones are transmitted. The signal matrix therefore has dimensions m-by-n equals 103-by-16. A moderately sized scan-grid, perhaps for a small home, consists of $G = 10,000$ points. The number of samples collected for each receive element is $N = 8192$. Using these

figures, the operation and data counts from above can be calculated. Table 2.6 summarizes these values, which are also represented visually in Figures 2.1 and 2.2.

Processing Stage Name	Operations [millions]	Data Out [MB]
Fast Fourier Transform	0.1065	0.0132
Rephasing	19.78	26.37
QR Decomposition	600.1	2.176
Bidiagonalization	131.1	0.496
Diagonalization	32.00	0.256

Table 2.1: Operation and data output counts for SART processing stages

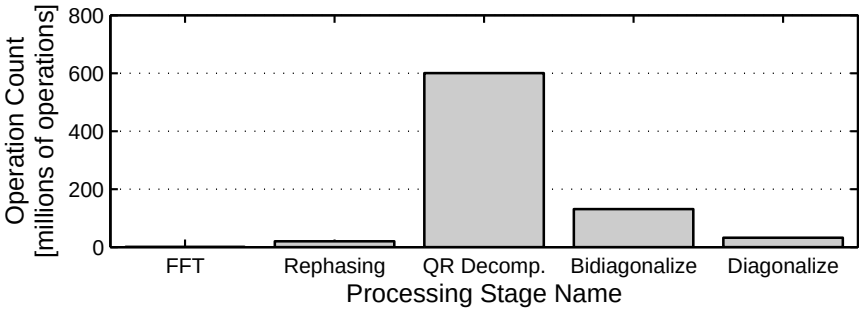

Figure 2.1: Operation count graph for SART processing stages

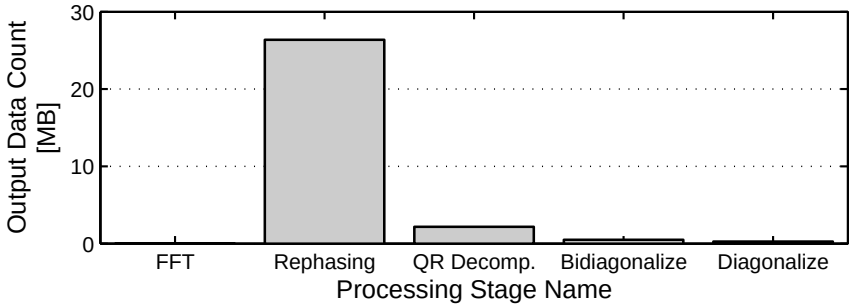

Figure 2.2: Data count graph for SART processing stages

It is clear that the processing associated with the FFT is negligible compared to the rest of the algorithm. This is mainly because FFT computations only happen once per SART scan, whereas all other operations happen once for each scan location. The rephasing stage also has a relatively low complexity. However rephasing generates a lot of output data, which could lead to data transfer bottlenecks. The QR decomposition stage presents the largest computational burden because of the large signal matrix height, m, and is an excellent candidate for accelertion. The bidiagonalization stage is less complex than the QR stage, but still has high relative complexity and is a good candidate for acceleration. The diagonalization stage has lower complexity, and a good speed up (of $600/30 = 20$ times) could be obtained even if its implementation remains in software.

These figures will be used in Section 4.2, when the SART algorithm is partitioned between hardware and software. The following chapter describes the hardware platform, which was selected for prototype development.

Chapter 3

System Platform

This chapter provides a description of the platform that was used to develop the SART co-processor system prototype. The benefits of field programmable gate array (FPGA) technology are discussed in general terms. The selected development platform is then presented, followed by more detailed description of the FPGA device on which the platform is based.

3.1 Field Programmable Gate Arrays

The highest performance digital signal processing systems available today are based on custom manufactured application-specific integrated circuits (ASICs). Photo lithographic manufacturing processes provide complete flexibility, allowing almost any digital circuit to be implemented using raw semi-conductor materials. These structures may be highly optimized so that very high clock rates, in the gigahertz, may be achieved. Unfortunately, ASIC manufacturing costs currently reach into the millions of dollars for complicated digital signal processing ICs [23].

Field programmable gate arrays (FPGAs) provide a lower cost solution for custom digital signal processing systems, and are often used for ASIC prototyping. FPGAs are produced using the same manufacturing process employed for application specific circuits, but are designed such that the function of the IC is not fixed. Each FPGA contains many small blocks

of digital logic that may be configured and interconnected by specifying the contents of the FPGAs configuration memory. In this way an FPGA provides a general purpose platform for constructing custom digital systems, which may consist of many sub-systems that operate in parallel. Unfortunately, FPGAs cannot achieve the same clock rates as ASICs; they are typically about 10 time slower [24]. This is because the routing network that provides interconnection amongst the elements within an FPGA is reconfigurable and not optimized like the routing within an ASIC. Furthermore, digital logic functions that would be implemented in an ASIC using logic gates must be implemented using small look-up tables (or memory elements) within an FPGA. Because these memory elements are slower than the equivalent combinations of gates, this also limits the clock rate of an FPGA implementation. Despite these limitations, FPGAs are widely used for implementation of digital signal processing systems, especially when the flexibility of a reconfigurable platform is advantageous. FPGAs are particularly popular in the production of specialized systems, when only a few units are produced and ASIC manufacturing costs cannot not be recouped.

3.2 Prototyping Platform

Although the final version of the SART co-processor will be implemented on a custom printed circuit board, an off-the-shelf development platform was selected for prototyping. The chosen platform is sold by Alpha Data, and is based on the Xilinx Virtex-4 SX55 FPGA. The SX55 is the largest in the SX series, which is geared towards digital signal processing applications. The Alpha Data platform is designed for compatibility with a standard PCI or PCI-X interface, and can be installed directly into a host PC. Alpha Data provides drivers and a programming interface that allow access to the FPGA using a memory-mapping scheme. In addition to the FPGA, the platform includes a PCI bridge for the host interface, and multiple banks of SRAM for data storage. A diagram of the platform is shown in Figure 3.1. The Alpha Data platform is relatively expensive, at about $5000. However, this is because it is general purpose, and contains components that will not be necessary in the final co-processor system. For example, the high-speed SRAM ICs cost a few hundred dollars each. Because the co-processor system was designed to minimize external memory requirements, this could be replaced with one inexpensive SDRAM IC.

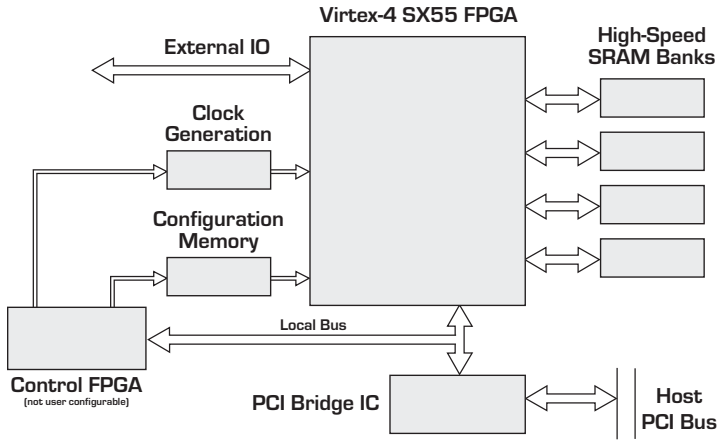

Figure 3.1: Diagram of the general purpose FPGA development platform from Alpha Data.

3.3 Virtex-4 FPGAs

When designing for an FPGA target, the architectural details of the device must be constantly considered, as they will heavily influence the final design. For example, the amount of memory available within the device will limit data storage and buffering, and the number of internal RAM banks will limit the rate at which internally buffered data can be accessed. The Virtex-4 series FPGAs are based on three types of building blocks: configurable logic blocks (CLBs), digital signal processing blocks (DSPBs), and memory blocks or "block RAM" (BRAM). In this section, these architectural elements are described. Various design challenges that arise due to the nature of these elements are noted along the way.

3.3.1 Configurable Logic Blocks

CLBs comprise the fundamental fabric of the FPGA. They can be used to implement all types of digital logic, including: logic gates, flip-flops, registers, state machines, and arithmetic functions. Each CLB contains four smaller blocks, known as "slices". Every slice contains two small 4-input look-up tables (LUTs), and two flip-flops. At runtime, the LUTs are configured in order to define their desired logic functions. For example, if a 4-input AND-gate is desired, then one LUT should be configured such that it outputs a logic-1 for an input consisting of four logic-1s, and a logic-0 for all other input combinations. The flip-flops may be used to register the output of the LUTs, or one of the slice inputs. Each

slice also contains additional logic for implementing a high-speed carry chain, which may be used in the implementation of binary adders or counters, and multiplexer circuitry for signal selection and dynamic routing. Multiple CLBs may be connected, using configurable routing resources, to form larger logic functions. The structures of a CLB and a logic slice are shown in Figure 3.2. The SX55 FPGA contains more than 55,000 CLBs.

In practice, the configurations of individual CLBs are almost never manually assigned. Instead, an engineer may use hardware description language (HDL) code to describe a digital circuit behaviorally or at a register transfer level. Powerful synthesis tools provided by the FPGA manufacturer, or third-party vendors, may be used to 'compile' the HDL code so that each function is automatically mapped into CLBs. However, despite this automation, the designer must always be considering the CLB architecture, and predicting how each line of code will be mapped to these functional blocks. For example, in an ASIC design, a 5-input AND-gate will have a propagation delay that is only fractionally larger than that of a 4-input gate. Whereas in an FPGA, the larger gate must be implemented using multiple LUTs, combined with multiplexer circuit to select one of the two LUT outputs. If more than two LUTs are needed for some logic function, then multiple slices or CLBs will be required, and additional routing delays must be considered.

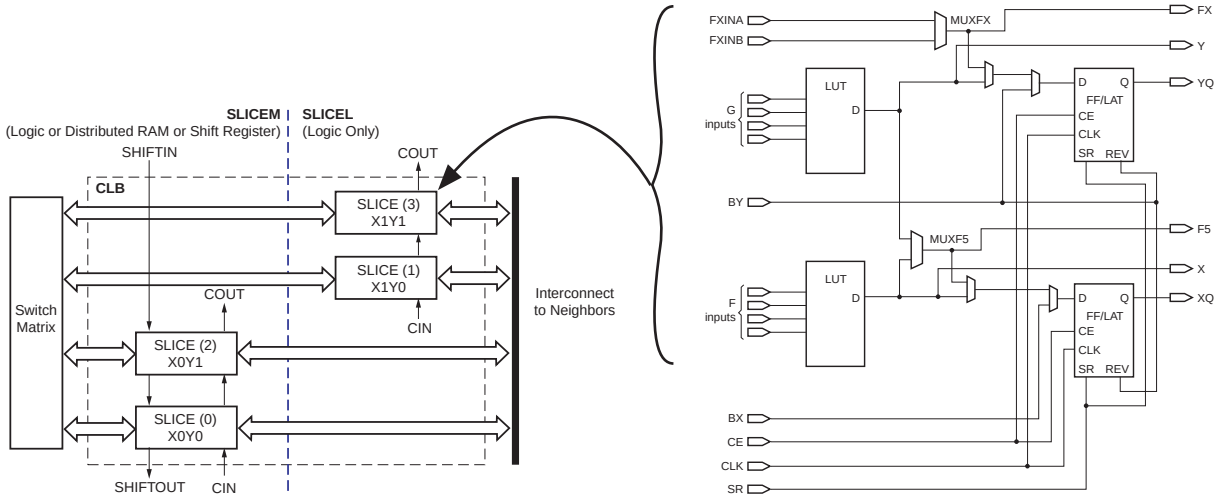

Figure 3.2: (a) Structure of a configurable logic block (CLB) [25]. Each CLB contains four logic "slices" which are connected to neighboring slices, and to the routing network through a reconfigurable switch matrix. (b) Structure of a logic slice. Each slice contains two 4-input look-up tables, two flip-flops (or latches), multiplexers, and other supporting logic.

3.3.2 Digital Signal Processing Blocks

FPGA based designs typically involve arithmetic operations. This is especially true for signal processing applications. For this reason, Virtex-4 FPGAs contain arithmetic circuit blocks, so that CLBs need not be used to implement arithmetic operations. These so called DSP blocks are highly optimized, and may be operated at higher clock rates compared to functionally equivalent CLB-based circuits. Each DSP block contains a 18-bit signed multiplier circuit, and a 48-bit adder/subtractor/accumulator circuit. Each circuit has a selectable output register, and various multiplexers and control lines are available for controlling the over-all behavior of the DSP block. Multiple DSP blocks may be cascaded to implement higher-precision operations. A simplified diagram of the Virtex-4 DSP block is shown in Figure 3.3. The SX55 contains 512 of these blocks, making it capable of highly-parallel signal processing.

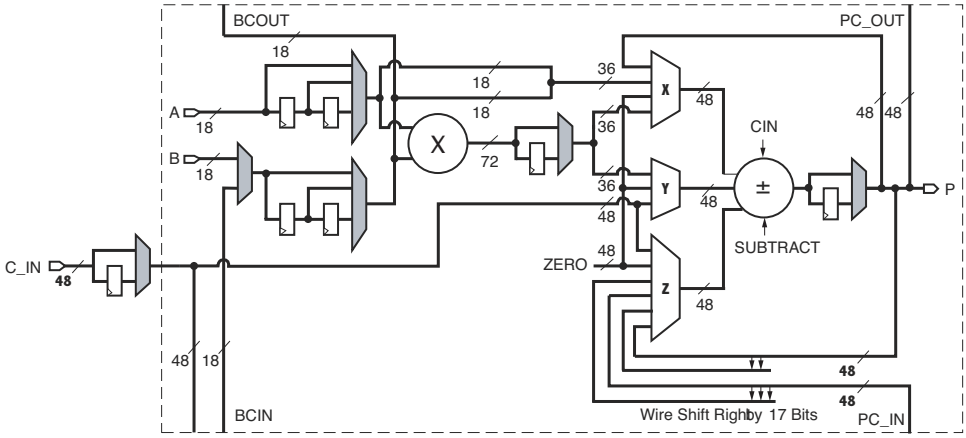

Figure 3.3: Structure of a DSP Block [26]

3.3.3 Block RAM

In order to allow storage and buffering of data, each Virtex-4 contains multiple blocks of dual-port SRAM. Each block has a capacity of 18kbits, and has configurable data port widths of up to 36 bits. The dual-port nature of the RAM allows simultaneous access to two memory locations, enabling the construction of first-in first-out (FIFO) type buffers, and shared memories. In fact, all necessary FIFO logic is included in each block so that CLBs need not be used. The SX55 contains 320 blocks of RAM, giving it a total of more than 5 Mbit of storage capacity, and an internal memory bandwidth greater than 9,000 Gbit/sec.

3.3.4 Resource Availability Constraints

The resources available in an FPGA are fixed in both quantity and location. This leads to an additional challenge when designing a digital system that targets an FPGA. The design must be a good match to the constraints imposed by resource abundance and distribution within the targeted device. For example, if a design targeted for the SX55 consists of two sub systems, one of which consists mostly of DSP blocks, and one that is composed purely of CLBs, then each subsystem is potentially wasting half of its available resources. This is because the DSP blocks are distributed amongst the CLBs, and a module consisting only of CLBs will potentially surround one or more DSP blocks, obscuring access to their ports. It is therefore beneficial to seek a match between the resource availability ratio, and the resource consumption ratio. Achieving this balance in the SX55 can be difficult because it contains many DSP blocks, and mapping functions into these blocks can be difficult and is not fully automated in the synthesis tools.

The SX55 was selected because of its suitability to high-performance DSP applications. However, due to tight constraints on CLB resources, this selection may not be completely appropriate. Section 7.1 contains a discussion about how other devices may be used in future versions of the SART accelerator. Table 3.1 contains information about the resource availability on various Virtex-4 devices. The current low-volume cost per device is also shown.

	Resources Available			
Device	DSP Blocks	CLBs	BRAM	Approx. Cost
LX60	64	60k	160	$600
LX80	80	80k	200	$1000
LX100	96	110k	240	$1600
LX160	96	152k	288	$3200
LX200	96	200k	336	$6000
SX35	192	35k	192	$550
SX55	512	55k	320	$1100
FX40	48	41k	144	$600
FX60	128	56k	232	$1000
FX100	160	94k	376	$2000
FX140	192	142k	552	$4800

Table 3.1: Available resources on various devices in the Xilinx Virtex-4 family FPGAs. Note: the FX series devices include additional features such as PowerPC processing cores, Ethernet MACs, and high-speed IO transceivers.

Chapter 4

SART Co-processor Architecture

The purpose of this work was to design a digital system capable of accelerating computations associated with the SART algorithm. The system needed to provided a solution that was compact and low-power compared to a cluster of general purpose computers. The accuracy of the output needed to be comparable to the existing solution, which employs single-precision floating point arithmetic. The system also needed to be scalable in order to accommodate larger signal matrices in the likely event that the number of receive elements or sub-carriers was increased. This chapter begins with a discussion of the design approach that was used to meet these goals. The SART algorithm is then partitioned into tasks suitable for hardware and software implementations. Remaining sections contain a description of the SART co-processor system, and all of its sub components.

4.1 Design Approach

The power and size requirements were satisfied by selecting FPGAs as the main processing elements. FPGAs consume less space and power than a PC with equivalent signal processing capabilities [24]. Like PCs, additional FPGAs may be used in parallel to obtain any processing performance required. An FPGA-based design also has the added benefit of being readily portable to an ASIC process [23]. Once optimized and implemented as an ASIC the design will run at clock frequencies up to 10 times higher, and consume even less power. Almost any

numerical precision can be implemented on an FPGA, so obtaining the required accuracy is possible. However, sometimes trade-offs between accuracy and resource consumption lead to compromise in this area. For example, FPGA designs often employ fixed point arithmetic because floating point operations require more hardware.

In order to obtain good scalability of the design, such that larger matrices could be processed by simply extending the same design, a modular approach was taken. The QR decomposition stage was implemented as a linear processing array (or systolic array), consisting of multiple identical array elements each of which processes only part of the input matrix. The size of the processing array many simply be increased for larger matrices. Because processing power scales up with the matrix size, array designs have speedup performance that increases with matrix size. The approach also promotes design reuse, which results in faster development and debugging [27].

In a processing array, data and control signals should flow from one array element to the next, being registered at least once in each element. This prevents "broadcasting", where a single signal must travel to many sub-systems during a single clock cycle. Because these sub-systems may reside on opposite sides of the FPGA, broadcasting uses more routing resources and results in lower clock frequencies. For the same reason, resource sharing should be limited to between adjacent modules. Limiting resource sharing also helps to avoid resource consumption associated with the implementation of large multiplexers.

Additional design goals were adopted based on the limitations and characteristics of the chosen FPGA. Within the SX55, there is only a limited amount of RAM. The memory available to each module was limited to a fraction of that amount. Similarly, the SX55 has many DSP blocks, and fewer CLBs than other FPGAs, so each processing element was designed to obtain most of its functionality from DSP blocks.

4.2 Algorithm Partitioning

The SART algorithm was partitioned such that a portion of the associated computations are conducted using custom co-processor hardware, while the remaining portion are conducted in software on the host PC. Three factors were considered when choosing the partition bound-

aries:

- the number of computations associated with each processing stage
- the amount of data flowing in and out of each stage
- the suitability to parallel implementation of each stage

The first factor is critical because the number of operations assigned to the host PC must be small, so that a sufficient speed-up may be obtained. The amount of data flow between stages is an important consideration because data that flows between the host and the co-processor should be limited. This will help to avoid data transfer bottlenecks, and minimize the number of memory transactions in the host so that its processing is not impaired. Though difficult to quantify, the suitability of each processing stage to parallel implementation is an important metric because implementation of highly sequential algorithms is typically less efficient and more difficult on a parallel platform such as an FPGA.

Using the results from Section 2.6, it was clear that the QR decomposition and bidiagonalization stages represent the bulk of the SART computational burden, and were therefore the prime targets for hardware implementation. The rephasing stage, though not computationally intensive, produces many rephased signal matrices for each input signal matrix. In order to reduce the amount of data transfered from the host to the co-processor system, the rephasing stage was also selected for hardware implementation. The remaining portions of the SART algorithm were assigned to the host PC, but could so be assigned to a general purpose DSP type processor. The FFT stage was assigned to the host in order to allow for manipulation of the frequency domain signal data before SART processing. This allows for various calibration and synchronization corrections to be applied by the host. The diagonalization stage was assigned to the host PC because it involves algorithms that are both sequential and iterative, and therefore better suited to a sequential processor. Using this partitioning, the number of operations assigned to the host PC was reduced by more than 95%. This corresponds to a potential speed-up of more than 20, if the co-processor performance is sufficient.

4.3 Top-Level Architecture

The main components of the SART co-processor system are the:

- Host PC interface
- Rephasing stage
- QR decomposition stage
- Bidiagonalization stage

The host interface allows the user to load configuration information and signal data into the co-processor. A memory mapping scheme is used, which allows access to the co-processor from any C or MATLAB program. The rephasing stage uses the data from the user to generate rephased signal matrices for all points on the chosen scan grid. These matrices are passed through the QR decomposition stage, which has been implemented as a linear processing array. Results from the QR decomposition stage are passed to the bidiagonalization stage. The bidiagonalization stage has been implemented as multiple identical processing modules, which work in parallel, but independently.

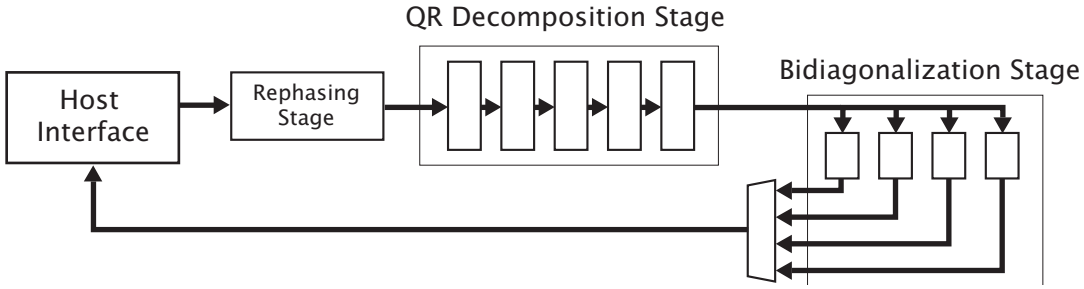

Figure 4.1: Top-level diagram of the SART co-processor system

The system description will begin with a discussion of the two fundamental operations that must be performed by the co-processor. This is followed by descriptions of the two processing modules, the "vector processing unit (VPU)" and the CODRIC module, which are used repeatedly throughout the system to implement theses fundamental operations. Subsequent sections will provide descriptions of the four sub-systems listed above.

4.4 Fundamental Operations

QR decomposition and bidiagonalization are accomplished by repeatedly performing Givens rotations in order to zero out all but the diagonal and super-diagonal elements of the complex signal matrix. The equations that a describe a Givens rotations involving two matrix row vectors, $\vec{a_j}$ and $\vec{a_k}$, that annihilates the j^{th} element in row k, a_{kj}, are:

$$\phi_j = angle(a_{jj}) \tag{4.1}$$

$$\phi_k = angle(a_{kj}) \tag{4.2}$$

$$\theta = tan^{-1}(\frac{-|a_{kj}|}{|a_{jj}|}) \tag{4.3}$$

$$\vec{a_j}'' = \cos(\theta)e^{-j\phi_j}\vec{a_j} - \sin(\theta)e^{-j\phi_k}\vec{a_k} \tag{4.4}$$

$$\vec{a_k}'' = \cos(\theta)e^{-j\phi_k}\vec{a_k} + \sin(\theta)e^{-j\phi_j}\vec{a_j} \tag{4.5}$$

These may be broken down and rearranged into an equivalent set of operations:

$$u_j = \frac{a_{jj}^*}{|a_{jj}|} \tag{4.6}$$

$$u_k = \frac{a_{kj}^*}{|a_{kj}|} \tag{4.7}$$

$$u_0 = \frac{|a_{jj}| - j|a_{kj}|}{\sqrt{|a_{jj}|^2 + |a_{kj}|^2}} \tag{4.8}$$

$$\Re\{\vec{a_j}'\} = \Re\{u_j\}\Re\{\vec{a_j}\} - \Im\{u_j\}\Im\{\vec{a_j}\} \tag{4.9}$$

$$\Im\{\vec{a_j}'\} = \Re\{u_j\}\Im\{\vec{a_j}\} + \Im\{u_j\}\Re\{\vec{a_j}\} \tag{4.10}$$

$$\Re\{\vec{a_k}'\} = \Re\{u_k\}\Re\{\vec{a_k}\} - \Im\{u_k\}\Im\{\vec{a_k}\} \tag{4.11}$$
$$\Im\{\vec{a_k}'\} = \Re\{u_k\}\Im\{\vec{a_k}\} + \Im\{u_k\}\Re\{\vec{a_k}\} \tag{4.12}$$

$$\Re\{\vec{a_j}''\} = \Re\{u_0\}\Re\{\vec{a_j}'\} - \Im\{u_0\}\Re\{\vec{a_k}'\} \tag{4.13}$$
$$\Im\{\vec{a_j}''\} = \Re\{u_0\}\Im\{\vec{a_j}'\} - \Im\{u_0\}\Im\{\vec{a_k}'\} \tag{4.14}$$

$$\Re\{\vec{a_k}''\} = \Re\{u_0\}\Re\{\vec{a_k}'\} + \Im\{u_0\}\Re\{\vec{a_j}'\} \tag{4.15}$$
$$\Im\{\vec{a_k}''\} = \Re\{u_0\}\Im\{\vec{a_k}'\} + \Im\{u_0\}\Im\{\vec{a_j}'\} \tag{4.16}$$

where a^* represents the complex conjugate of a, $\Re\{a\}$ represents the real part of a, and $\Im\{a\}$ represents the imaginary part of a.

In this form, it is clear that there are two fundamental operations required to achieve a Givens rotation. Firstly, as described by Equations 4.6, 4.7, and 4.8, computation of unit vectors must be performed. Second, apart from a few sign changes, Equations 4.9 through 4.16 have identical 'sum-of-products' forms, so this operation also must be implemented. Because these two operations are so fundamental to the SART co-processor design, the following sections describe how they were mapped into the architecture of the SX55 FPGA.

4.5 Vector Processing Unit

The vector rotation operations described by Equations 4.9 through 4.11 were implemented in a module which has been labeled the vector processing unit (VPU). The module was constructed using 16 DSP blocks, and performs four 35-bit multiplications and two 35-bit addition/subtractions in parallel in order to produce a result for two of the equations simultaneously. It is completely pipelined to provide a throughput of one result per clock cycle after a latency of 10 cycles. An opcode input is used to select which two operations are to be performed on the current set of input vectors.

Given the inputs:

- $a + jb$
- $c + jd$
- $cos(\theta) + jsin(\theta)$

the VPU can produce the following outputs:

- $x = acos(\theta) - bsin(\theta)$ [Equation 4.9 or Equation 4.11]
  $y = bcos(\theta) + asin(\theta)$ [Equation 4.10 or Equation 4.12]
- $x = acos(\theta) - csin(\theta)$ [Equation 4.13]
  $y = bcos(\theta) - dsin(\theta)$ [Equation 4.14]
- $x = ccos(\theta) + asin(\theta)$ [Equation 4.15]
  $y = dcos(\theta) + bsin(\theta)$ [Equation 4.16]

The equations have the same basic form involving two multiplications, whose products are either summed or subtracted. This operation was implemented as a "multiply-add" sub-module, which is described in the next section. As shown in Figure 4.2, each vector processing unit contains two of these multiply-add modules.

One multiply-add module computes the real component of the result, while the other computes the imaginary component. The inputs to the modules are selected from the six inputs, based on the operation input code, *op*. The op-code is also used to generate the $sub1$ and $sub2$ signals, which control the product sign inversions. The valid-input, *vin*, and operation code, *op*, signals are delayed according to the latency of the multiply-add module, generating valid-out, one-cycle-early valid-out, operation-out, and one-cycle-early operation-out signals, which are used to propagate control information.

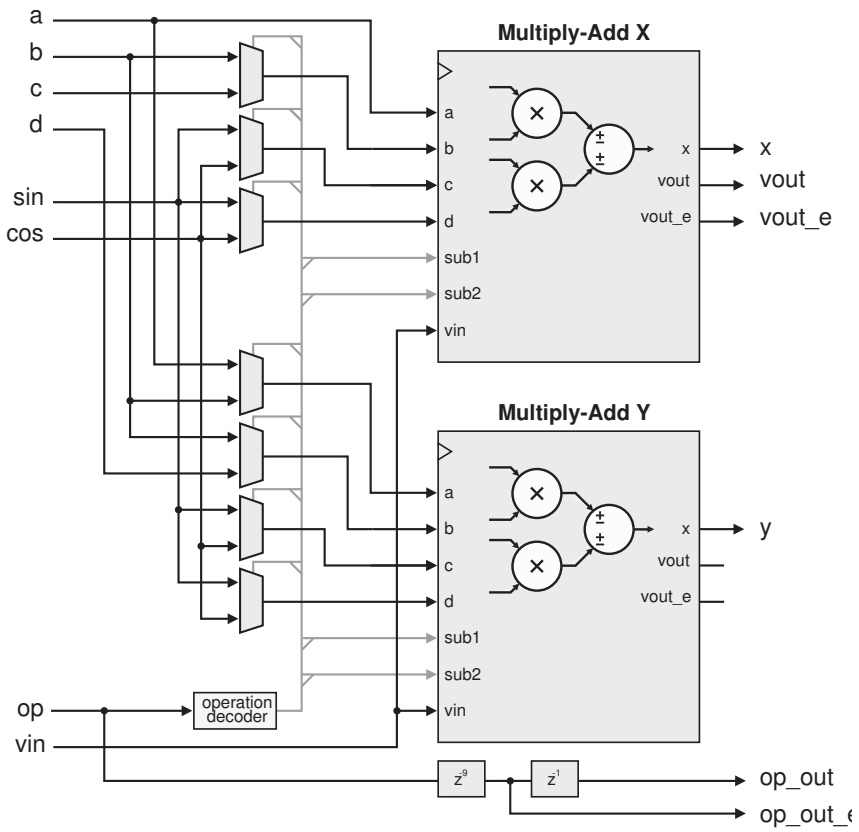

Figure 4.2: Vector Processing unit consisting of two multiply-add modules, operation decoder, and input multiplexers

4.5.1 Multiply-Add Module

The multiply-add module is used repeatedly throughout the SART co-processor architecture. It accepts four 35-bit, two's-complement formatted input values, a, b, c, and d, and two 1-bit sign values, $sub1$ and $sub2$. From these values, the circuit computes two products which are then summed with optional sign inversions, specified using the two sign bits. This operation is described in Equation 4.17.

$$x(a, b, c, d, sub1, sub2) = ac(-1)^{sub1} + bd(-1)^{sub2} \tag{4.17}$$

A simple, high-level diagram illustrating this function as a parallel system is shown in Figure 4.3. Two modules that perform multiplication are connected to two adder/subtracter modules. A register stores the output of each module to reflect the normal pipelining method for achieving maximum clock frequencies. This system would produce a new result on each

clock cycle, and have a latency of 3 clock cycles.

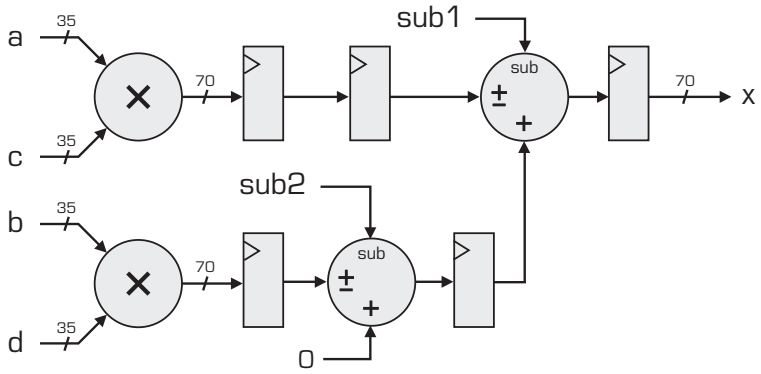

Figure 4.3: Diagram of a simple parallel system for computing a sum of two products

Because of the restrictions imposed by the architecture of the SX55 FPGA, the system depicted in Figure 4.3 can not be implemented as a circuit directly. The sum of products function must be mapped into the target architecture such that it makes efficient use of the available resources. The SX55 FPGA is intended for signal processing applications, and therefore contains many DSP blocks that are optimized for arithmetic operations. The structure of a DSP block is shown in Figure 3.3. The module contains an 18-bit signed multiplier circuit, and a 48-bit adder/subtractor/accumulator circuit. Therefore, the 35-bit multiplication implemented in the multiply-add module cannot be implemented using a single block.

In order to obtain higher precision, multiple DSP blocks must be combined. As shown in Figure 3.3, each DSP block includes auxiliary outputs and inputs, PC_OUT and PC_IN, which allow results from one DSP block to be passed to an adjacent DSP block where the value may be added to the multiplier output in that block. Furthermore, the PC_IN value may be optionally right-shifted by 17 bits. This allows inputs corresponding to lower significant digits to be appropriately adjusted in magnitude. For example, in order to multiply two 35-bit numbers using 18-bit operations, the high and low words from one operand must each be multiplied by the high and low words of the remaining operand. The products corresponding to the multiplication of a high word with a low word must be right shifted by 17 bits. The product of the two low words must be right shifted by 34 bits. The four partial products may then be added to obtain the final result of the 35-bit multiplication. This is illustrated in Figure 4.4.

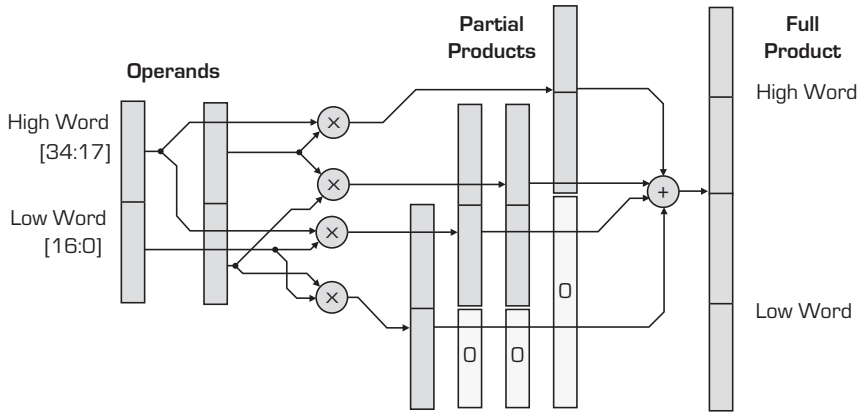

Figure 4.4: A 35-bit multiplication can be implemented using 18-bit multipliers by generating 18-bit partial products, which can be shifted and summed to obtain the full product

Using this technique, four DSP blocks can be used to implement each multiply operation in the multiply-add module, as shown in Figure 4.5. Each of the four DSP blocks computes a partial product, which is passed vertically using the PC_IN and _OUT ports, shifted if necessary, and added to the other partial products. The first partial product is computed after a 3 cycle latency in DSP block 0. The inputs to DSP block 1 are delayed by an additional cycle, such that the second partial product arrives at the adder in block 1 during the same cycle as the first partial product. The inputs to DSP blocks 2 and 3 are delayed by two and three cycles, respectively, for the same reason. The result of the multiplication is available six cycles after the operands are presented to the circuit's input registers. The circuit is fully pipelined to generates one new result per clock cycle.

In order to implement the sum of products operation, two of these multiplier circuits could be connected to another DSP block that performs the addition operation. However, use of an additional DSP block can be avoided by interleaving two 35-bit multipliers, and performing the addition as part of the partial product summation. The least significant partial products must be computed at the bottom of the DSP block chain, so they may be properly shifted as they propagate upwards to be combined with partial products of higher significance. The final 4-input multiply-add circuit is shown in Figure 4.6. The latency of the circuit is 10 cycles.

The presented configuration also provides the required sign inversion capability. Each DSP slice in the FPGA includes a control bit that allows for selection of a subtraction operation instead of addition. In this design, the subtraction control bits for the blocks that perform

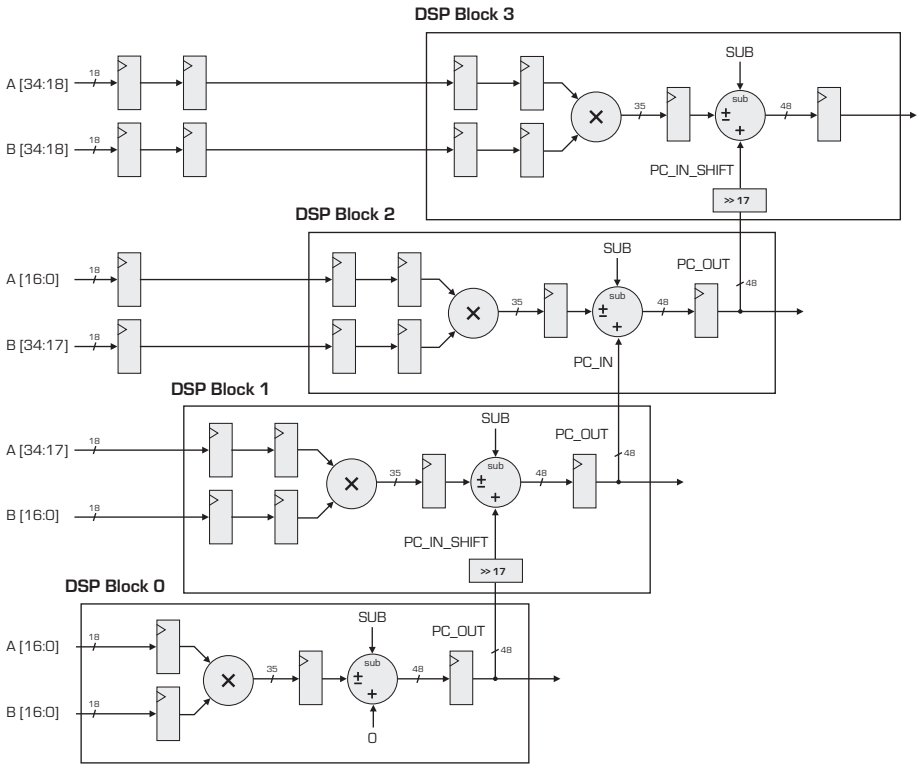

Figure 4.5: 35-bit multiplication mapped to four DSP blocks

multiplication of inputs a and c are connected to the s_1 input signal. The DSP blocks that generate the product of b and d are connected to the s_2 control bit. In each case, the control signals are delayed to match the pipeline latencies at each DSP block. By inverting the individual partial products, either of the full products may be inverted during the summation.

The multiplication of two 35-bit values produces a 70-bit result. However, because only the top 35-bits are desired, the result must be rounded. This is accomplished by adding 1/2 of an LSB (least significant bit with respect to the desired 35-bit value) to the 70-bit result, and then truncating. This rounding constant is applied to the C_IN port of DSP block 0 and included in the sum of partial products. The final sum-of-products result is truncated to 35-bits.

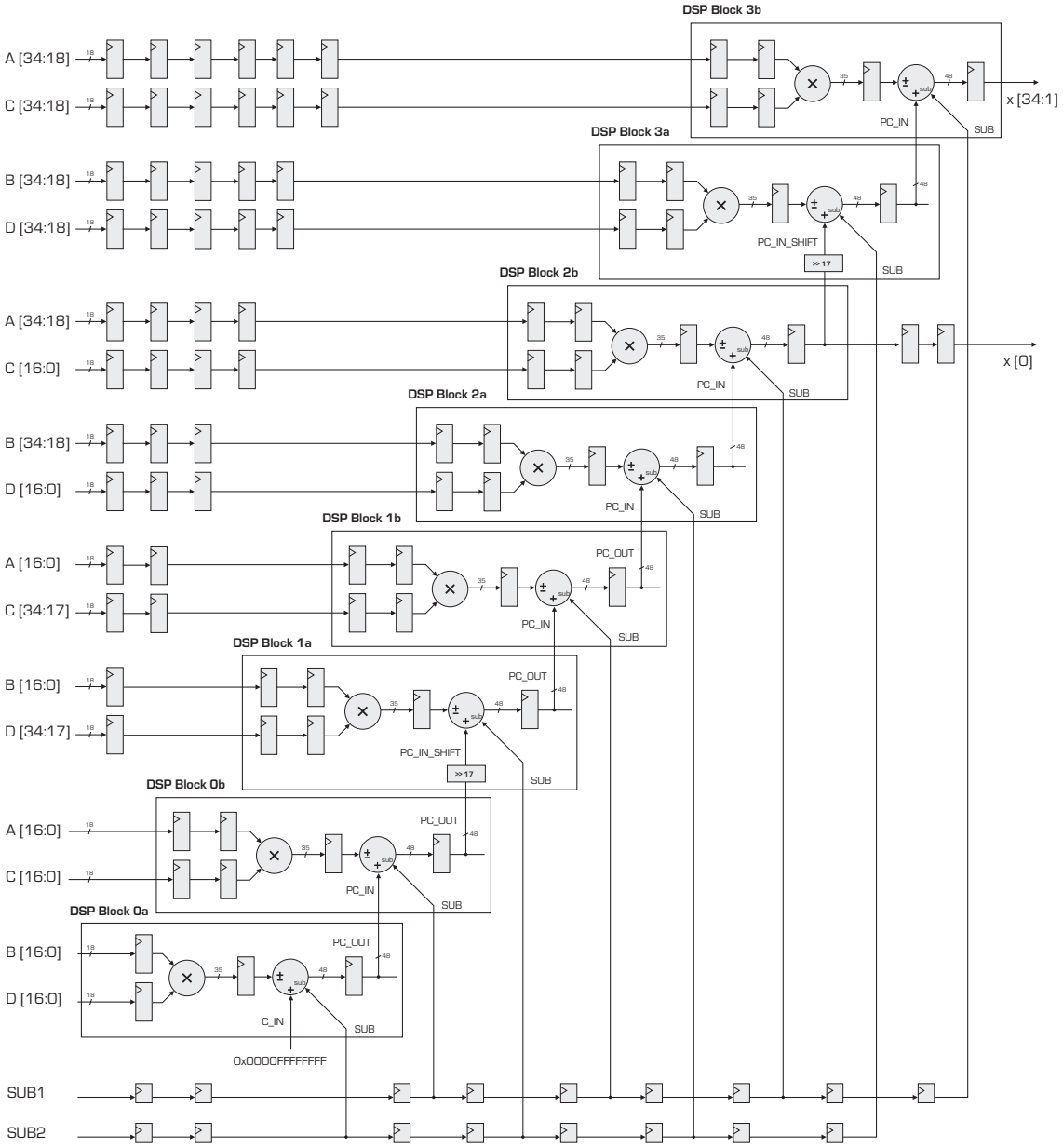

Figure 4.6: 35-bit multiply-add mapped to eight DSP blocks

4.6 CORDIC Module

The COrdinate Rotation Digital Computer (CORDIC) algorithm may be used to implement vector rotations and related trigonometric functions. Using the CORDIC technique, vector rotations are computed using only shift, and add operations. That is, if the rotation angle, θ, is chosen such that $tan(\theta) = 2^{-n}$, where n is a positive integer, then a vector rotation operation can be reduced to the form shown in Equations 4.18 and 4.19.

$$kx' = x \mp y2^{-n} \tag{4.18}$$

$$ky' = y \pm x2^{-n} \tag{4.19}$$

This simplificaiton is obtained at the expense of an introduced gain factor, k, which can be compensated for at the end of series of these rotations, by multiplying by the inverse gain factor, $K(p)$, shown in Equation 4.20

$$K(p) = \prod_{i=0}^{p-1} cos(atan(2^{-i})) \tag{4.20}$$

where p is the number of compounded rotation operations, and usually equal to numerical precision of the arithmetic. For rotations vectors beyond the first quadrant, coarse rotations of 90, 180, and 270 degrees can be performed by manipulating the signs of the real and imaginary vector components and/or swapping these components before CORDIC processing.

In the context of the CORDIC algorithm, the operation described by Equations 4.18 and 4.19 is known as a micro-rotation. By applying a sequence of these micro-rotations, with each rotation angle smaller than that of the previous rotation, any rotation angle may be achieved. When the CORDIC algorithm is used for rectangular to polar conversion, each rotation is chosen to be in either the positive or negative direction such that the y-component of the vector is reduced in magnitude, and the corresponding rotation angles (stored in a small look-up table) are accumulated. If the desired operation is vector rotation by an arbitrary angle, then this angle can be converted to a boolean vector corresponding to the rotation direction for each micro-rotation. If a unit vector is desired, two CORDIC rotations may be performed together, with the second rotation sequence mimicking the first. In this way, a unit vector lying along the x-axis may be rotated to an orientation that is parallel to the conjugate of the input vector. Furthermore, the unit vector may be pre-scaled by $K(p)$ in order to eliminate

the gain compensation step at the end of the rotation. Unit vector calculation using this technique is illustrated in Figure 4.7. In any case, the CORDIC algorithm converges to a machine precision, of p bits, after p rotations.

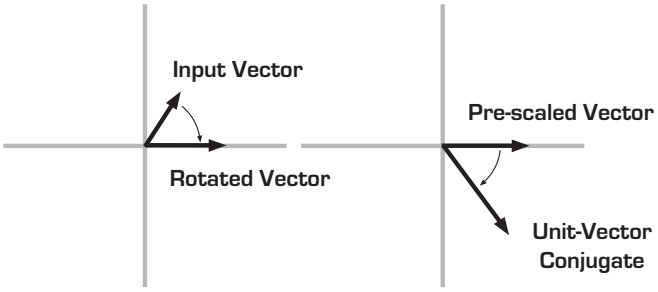

Figure 4.7: Generation of a unit vector using coordinated CORDIC rotations

4.6.1 Implementation Considerations

Xilinx, the manufacturer of the SX55 FPGA, includes a CORDIC module in their freely-available design library. Unfortunately, this implementation was not well suited to the SART co-processor system. As will be discussed in Section 4.9, the QR decomposition stage was implemented as a linear processing array, which is intended to be very modular. Each element in the array requires the ability to perform two CORDIC rotations simultaneously. It was found that both CORDIC implementations provided by Xilinx were too large for the modular approach. The sequential algorithm implementation, which uses one circuit to perform all micro rotations in sequence, occupied too many of the CLB resources allotted to one array element, as shown in Figure 4.8. The parallel implementation, which uses a pipeline with P stages to perform P rotations simultaneously, was far to large to fit inside a single module. The parallel implementation may have been appropriate if resource sharing was employed, but this would have violated the modular design approach, and complicated the overall design.

The main problem with the Xilinx CORDIC module is due to the fact that it is constructed from configurable logic blocks only. No DSP blocks are employed. This makes sense in the context of many systems, especially when remembering that the CORDIC algorithm was designed to avoid multiplication operations. Because each DSP slice contains a multiplier, it may seem wasteful to implement CORDIC using them. However, sequential CORDIC

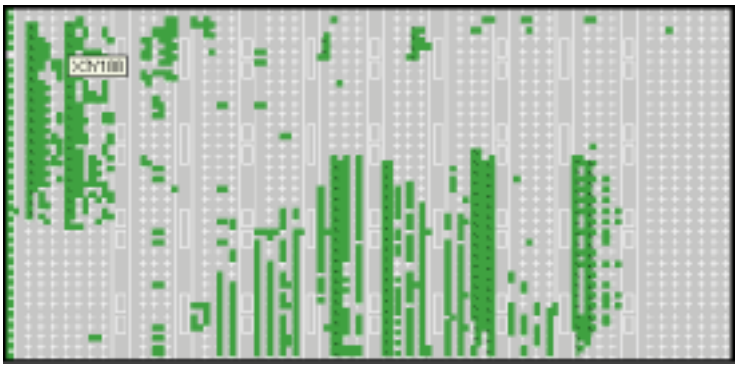

Figure 4.8: The Xilinx CORDIC implementation consumes a large portion (dark regions) of the total resources allotted to one element in the QR decomposition processing array (rectangle).

algorithms can be difficult to implement efficiently because each micro-rotation involves a shift-add operation. In a parallel implementation, the output of one stage is may simply be connected to the input of the next stage such that the shift is implemented using wires. In a sequential CORDIC implementation, however, a shift-by-n circuit must be constructed. If the shift is to be performed in one clock cycle, then this circuit becomes quite complex. This may account for the large size of the Xilinx module.

The design presented here employs the DSP-block multiplication circuit to perform the shift-by-n operation, because the shift operation is essentially a multiplication by a power of two. In order to obtain the desired precision, multiple DSP slices must be cascaded as for the multiply-add circuit in Section 4.5. Figure 4.5 shows a 35-bit multiplier structure. Unfortunately, some challenges arise due to the pipelining of the DSP blocks, and the data dependency in the CORDIC algorithm. That is, for each successive micro-rotation, a decision about rotation direction must be made. This decision is made based on the sign of the y-component of the rotated vector, which is dependent on the previous micro-rotation. Therefore, a result from one rotation must exit the pipeline, before the data for the next rotation may enter the pipeline. In order to use the pipeline efficiently, multiple CORDIC rotations may be performed simultaneously, such that the pipeline is fully occupied.

4.6.2 CORDIC Implementation

Implementation of the CORDIC module began with the design of a shift-accumulate circuit using DSP blocks. As noted, a shift operation may be viewed as a multiplication operation, where one of the operands is a power of two. The four-block multiplier shown in Figure 4.5 was simplified to a shifter composed of two DSP blocks. This is possible because the operand that is a power of two contains only one bit that is set, and all other bits are zeros. Therefore, either the high word, or low word of this operand is zero, and need not be factored into the calculation. Multiplexers were used to select the appropriate word from the operand to be shifted. The accumulation defined by Equations 4.18 and 4.19 was accomplished by connecting the output of the shifter to the C_IN port of the bottom DSP block. This feedback value is shifted to the left by 17 bits in order to compensate for the left-shift that occurs between the lower and upper DSP blocks. A register was inserted into the feedback path to add one cycle of delay. This results in a match between the latency of the shift operation and the latency in the feedback propagation. The subtraction control line allows the shifted value to be either added or subtracted from the accumulated value. The shift-accumulate circuit is shown in Figure 4.9.

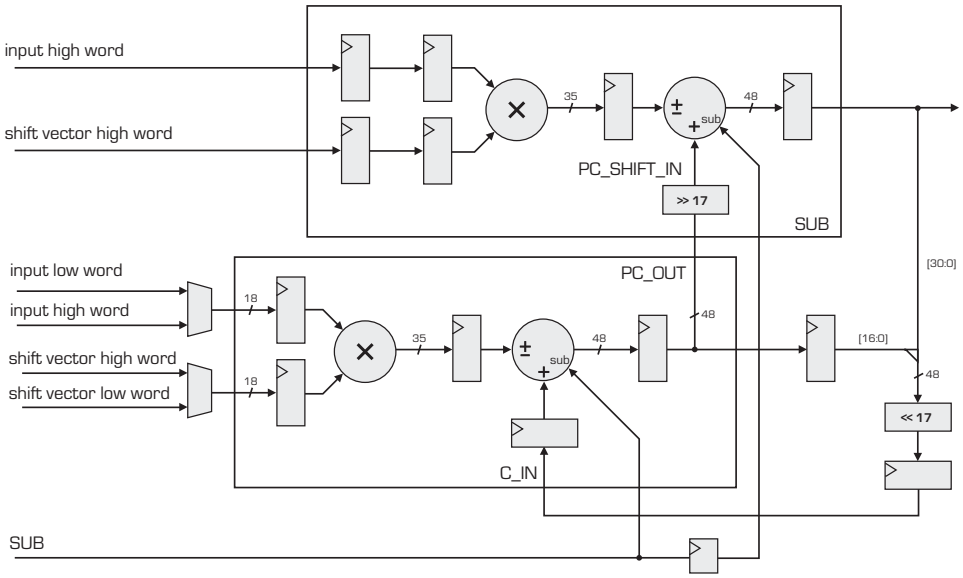

Figure 4.9: CORDIC shift-accumulate operation implemented using two DSP blocks.

In order to implement a CORDIC operation, two shift-accumulate circuits were combined. One serves to compute the x-component (real component) of the rotated vector, while the

other computes the y-component (imaginary component). For coarse vector rotation, which may involve swapping of the x and y components, multiplexers were added to the feedback paths in the shift-accumulate circuits. One multiplexer input maintains the feedback loop, while the second allows selection of the output of the other shift-add circuit. Using the multiplexer control line, either feedback or swapping may be selected. The circuit is shown in Figure 4.10.

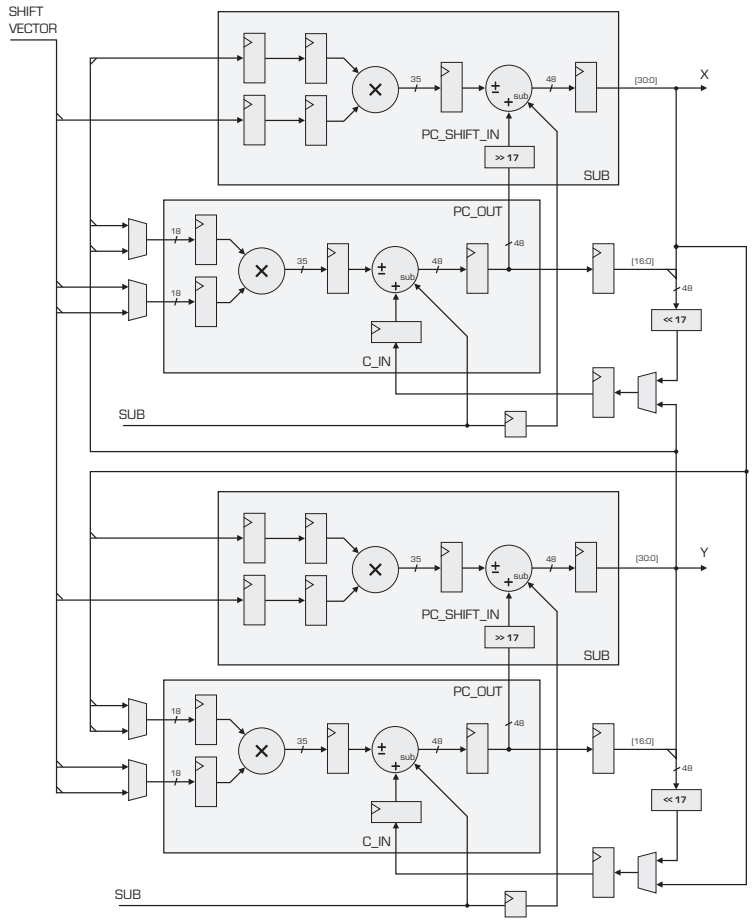

Figure 4.10: CORDIC rotation circuit implemented using two shift-accumulate circuits.

The latency of the CORDIC circuit is four cycles. In order to use the circuit efficiently, the pipeline should be filled completely. This means that the circuit will compute four CORDIC rotations simultaneously. In order to supply these inputs, some form of input multiplexing was necessary. In order to save on CLB and routing resource consumption, additional DSP slices were used as large multiplexing circuits, as shown in Figure 4.11. The inputs to the CORDIC circuit are connected to these blocks. When the CORDIC process is triggered, these

values are shifted up and into the CORDIC processing circuitry. This is accomplished by selecting the PC_IN input instead of the C_IN input on the bottom DSP block in the CORDIC circuit. After the values are loaded, the circular pipeline is closed by switching back to the C_IN input and processing begins. The subtraction control lines on the multiplexer DSP block are used to perform sign inversion required for coarse rotation at the beginning of the CORDIC rotation procedure.

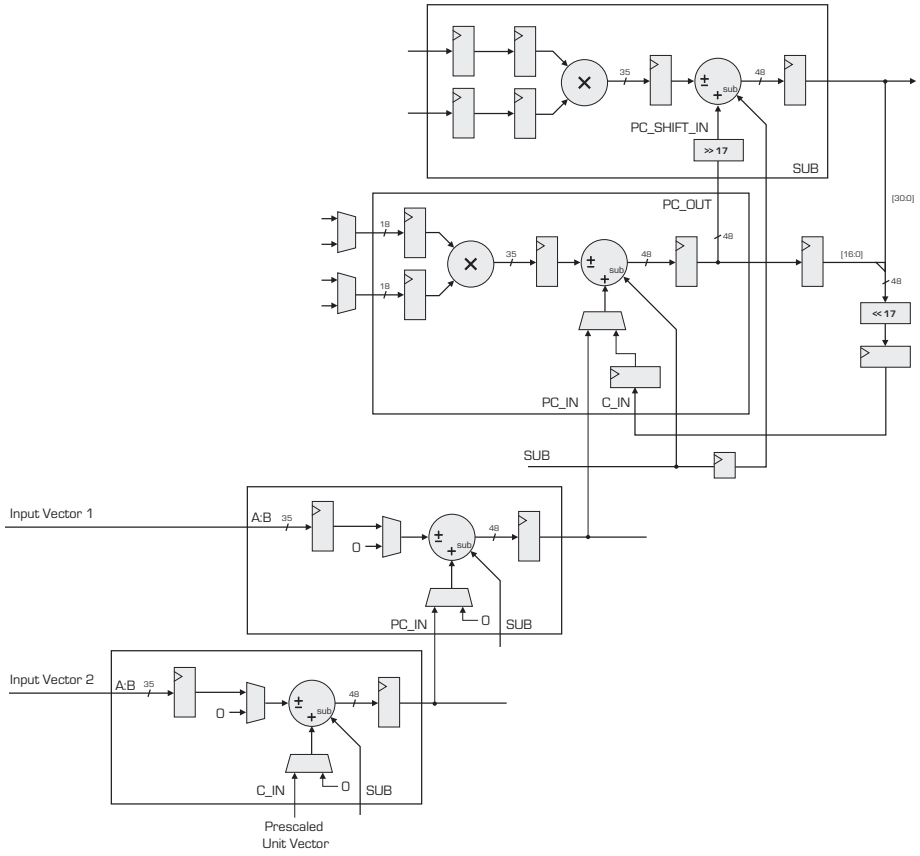

Figure 4.11: Inputs to the CORDIC circuit are selected using a wide multiplexer implemented with two DSP slices. Inputs are connected to ports A and B, which are internally concatenated. The operand multiplexers within the DSP slice allow selection of either the input or zero, such that only one non-zero input is passed to the adder circuit. Inputs are shifted up into the CORDIC circuit sequentially. The bottom DSP slices is also used to shift a pre-scaled unit vector into the circuit, as in suggested in Figure 4.7

As will be discussed in the QR decomposition section, each CORDIC circuit performs computations on two input vectors. For both input vectors, the circuit calculates a unit vector with a phase angle equal to that of the complex conjugate of the input vector. For the first input vector, the circuit also calculates the vector magnitude. Two of the four pipeline slots are used for rotating the input vectors to a phase angle of zero. The other two pipeline slots compute the unit vectors by mimicking theses rotations, as depicted in Figure 4.7. At the end of the micro-rotation sequence, the scaling factor from Equation 4.20 is applied to the first vector, which now lies along the x-axis. Compensation for the CORDIC gain factor is necessary for obtaining the magnitude of the first input vector. The scaling factor is applied using the DSP blocks, in the same manner as a shift operation, but requires two pipe cycles because both the high and low words of the scale factor operand contain non-zero bits.

The shift vector input to the CORDIC circuit, which is either a power of two or the aforementioned scale-factor, is stored in a block RAM (BRAM) module. An address counter connected to the BRAM increments on every cycle after the CORDIC computation has been triggered. This BRAM is also used to store the control signals that dictate the operation modes of the four DSP slices on each clock cycle. The processing schedule for the CORDIC module is shown in Figure 4.13.

By mapping the CORDIC algorithm into DSP blocks as described here, the footprint of the CORDIC module was greatly reduced. Figure 4.12 shows its footprint in relation to the size of one element in the QR decomposition array, so that it can be compared to the footprint of the Xilinx CORDIC module in Figure 4.8.

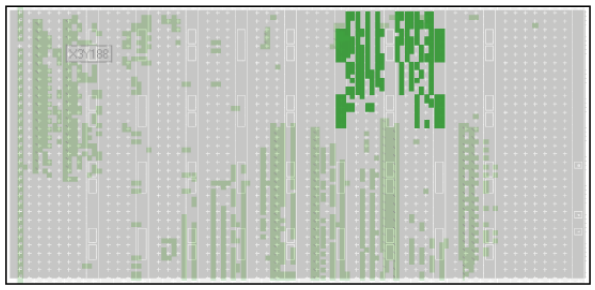

Figure 4.12: The footprint of the CORDIC implementation (dark regions) was reduced compared to the Xilinx implementation (lighter regions), and occupies only a small portion of the total area allotted to a single element in the QR decomposition processing array (rectangle).

Operation Perfomed on Current Cycle		Data Processed by DSP Block			
		Mux Block Bottom	Mux Block Top	Processing Block Bot.	Processing Block Top
	Loading	Unit Vect1	Input 1	x	x
		Input 2	Unit Vect1	Input 1	x
		Unit Vect 2	Input 2	Unit Vect 1	Input 1
		x	Unit Vect 2	Input 2	Unit Vect 1
	Swapping	x	x	Unit Vect 2	Input 2
		x	x	Input 1	Unit Vect 2
		x	x	Unit Vect 1	Input 1
		x	x	Input 2	Unit Vect 1
	Rotating	x	x	Unit Vect 2	Input 2
		x	x	Input 1	Unit Vect 2
		x	x	Unit Vect 1	Input 1
		x	x	Input 2	Unit Vect 1
	(Repeats 31 times)	...	...	...	...
	Scaling	x	x	Input 1	x
		x	x	x	Input 1
		x	x	x	x
		x	x	x	x
		x	x	Input 1	x
		x	x	x	Input 1

Figure 4.13: Processing schedule for CORDIC unit. The left column shows the operation begin performed. The remaining four columns show the sources of the data being processed by each DSP block in a shift-accumulate circuit.

4.7 Host Interface

The host PC interfaces with the SART co-processor system through a PCI bridge IC. Alpha-Data provides all logic descriptions and software necessary for supporting this interface, as a part of the prototyping platform development package. The included HDL files were used to instantiate a state machine and logic, inside the FPGA, for controlling the bridge IC via its "local bus" signals. These components reduce the interface to simple address and data buses, with additional signals to indicate read and write transactions, and acknowledge data availability. Memory and control elements within the FPGA are mapped into the memory space of the host PC using adding additional logic for decoding the value on the address bus, and placing output of the appropriate module on the data ouput bus. Device drivers for the Alpha-Data development platform may be installed on any PC that supports the PCI compatible interface. These drivers allow the included API libraries to be used in C programs or MATLAB scripts for controlling the SART co-processor system. Table 4.7 shows the memory mapping assignments for the SART co-processor system.

Table 4.1: Address ranges for various memory mapped system elements

Memory Element or Control Signal	Write	Read
Reset Signal	0x000000	none
Control and Status Register	0x000004	0x000004
SRAM - Rephasing Vectors	0x200000-0x3FFFFF	none
Signal Matrix - Real Component	0x040000-0x043FFC	none
Signal Matrix - Imaginary Component	0x080000-0x083FFC	none
Result Buffer	none	0x0C0000-0x1BFFFF
Bidiagonalize Stage Program RAM	0x1C0000-0x1C03FC	none

4.8 Rephasing Stage

Rephasing of the SART signal matrix is accomplished by element-wise multiplication with a "phase reference" matrix. The phase reference matrix is simply the set of complex exponentials that describe the frequency-dependent phase shifts for the current scan grid location (see Section 2.4). Since both the signal matrix and the phase reference matrix are com-

plex, their multiplication was easily implemented using the multiply-add module described in Section 4.5. However, storage of large size scan grids presented a problem due to memory constraints of the system. Even if all four banks of external SRAM were used, there would only be enough space for a scan grid with about 630 points. A custom system could certainly contain more memory, but an alternate solution was necessary for producing a useful prototype system.

4.8.1 Rephasing Matrix Compression

In order to reduce the amount of memory occupied by the SART phase reference data, a simple compression scheme was adopted. Because the current implementation employs a signal with sub-carrier tones that are placed on evenly-spaced frequency intervals, the phase shifts undergone by theses sub-carriers are also evenly spaced. For example, if the first sub-carrier undergoes a phase shift of θ_0, then the second and third sub-carriers with undergo phase shifts of $\theta_0 + \Delta\theta$ and $\theta_0 + 2\Delta\theta$. Furthermore, the initial phase shift, θ_0, does not effect the singular values of the signal matrix because it does not effect the linear dependence between the effected column and other columns in the signal matrix. Ignoring θ_0, the phase shift of the k^{th} sub-carrier may be expressed as

$$\Theta_k = k\Delta\theta \tag{4.21}$$

the phase reference vector that encodes this phase shift is

$$U_k = e^{jk\Delta\theta} = \prod_{i=0}^{k} e^{j\Delta\theta} \tag{4.22}$$

therefore the phase reference vector for any one sub-carrier may be obtained by multiplying the phase reference vector of the previous sub-carrier with the constant phase step vector, $e^{j\Delta\theta}$. This can be expressed as

$$U_k = U_{k-1}e^{j\Delta\theta}, \qquad where\ U_0 = 1 \tag{4.23}$$

By storing only this phase step, the amount of memory consumed by the phase reference array may reduced by a factor equal to the number of sub-carriers. For the current signal

matrix, the storage space is reduced by a factor of more than 100. Using this method, the current SART co-processor system supports a scan grids of up to 16,384 points with a single SRAM bank.

4.8.2 Input Interface

The implementation of the rephasing stage input interface is depicted in Figure 4.14. The host PC accesses the module using the PCI bus interface provided by Alpha Data. The signal matrix is loaded into to block RAM modules, which store the real and imaginary components of the data. This RAM is large enough to store two signal matrices, so that a new signal matrix may be tranfered from the host while the current signal matrix is being processed. The SART scan-grid rephasing matrices are written to external SRAM in the compressed format discussed in the previous section. For each location on the scan-grid, two 32-bit value are written for each column of the signal matrix. These values represent the real and imaginary components of the phase-step vector, $e^{j\Delta\theta}$, discussed in Section 4.8.1. On each clock cycle, the SRAM operation alternates between read and write so that the device may function as a shared resource. Values and memory addresses from the host are buffered so that they may be written to SRAM when the device is available. The interface to the SRAM IC also was provided by Alpha Data as part of the development platform support library.

4.8.3 Rephasing Matrix Decompression and Application

The rephasing matrices are decompressed, as they are needed, using the iterative technique described in Equation 4.23. One phase step vector, $e^{j\Delta\theta}$, is retrieved from SRAM for each column of the matrix, and stored in a pair of FIFOs buffers. An equal number of unit length, zero-phase vectors are loaded into a second pair of FIFOs, these are the rephasing vectors for the first row in the signal matrix. In subsequent cycles, calculation of the rephased signal matrix and decompression of the rephasing matrix happen simultaneously, with one element of the rephased signal matrix and one element of the rephasing matrix being generated on each clock cycle.

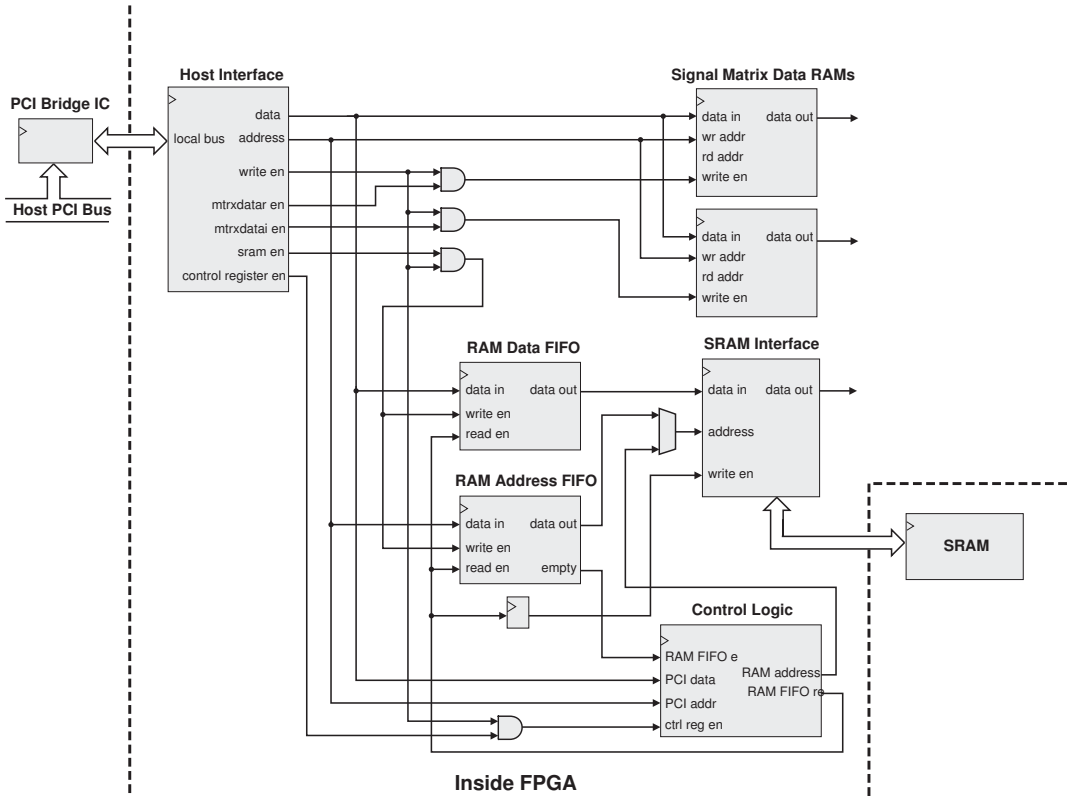

Figure 4.14: Rephasing stage input interface

The FIFOs that hold the real and imaginary components of the phase step vectors are read on each clock cycle. Their outputs are fed back to their inputs in order to form cicular buffers that will repeat their output once for each row in the signal matrix. As each phase step vector appears on the outputs of the circular buffers, it is multiplied by the rephasing vector for the previous row, thus generating the rephasing vector for the current row. Each newly generated rephasing vector is applied to the appropriate element in the signal matrix using a second complex multiplier, and also passed to the input of the rephasing vector buffer for use during the next phase step iteration.

The signal matrix is processed beginning with the bottom row and first column. On each clock cycle the column number is advanced by one. After the last element in the current row has been processed, then the row number is decremented, such that processing of the next row up may begin. The rephased signal matrix is stored in a FIFO buffer so it may be used when the QR decomposition stage is ready for new data. A diagram of the rephasing processing circuit is shown in Figure 4.15. It is pipelined such that an m-by-n matrix may

be rephased in mn cycles after n phase step vectors have been retrieved from RAM.

The control logic block shown in Figure 4.15 consists of a set of counters and registers. Two counters are used to keep track of which row and column are currently being processed. A third counter is used to keep track of which scan-grid point is currently being processed. Processing occurs in bursts, such that multiple scan-grid points are processed before the host must retrieve the computation results. The host system may load a control register with the start and end points for the burst. The maximum burst length is 128 scan grid points, and is limited by the size of the output buffers in the bidiagonalization stage.

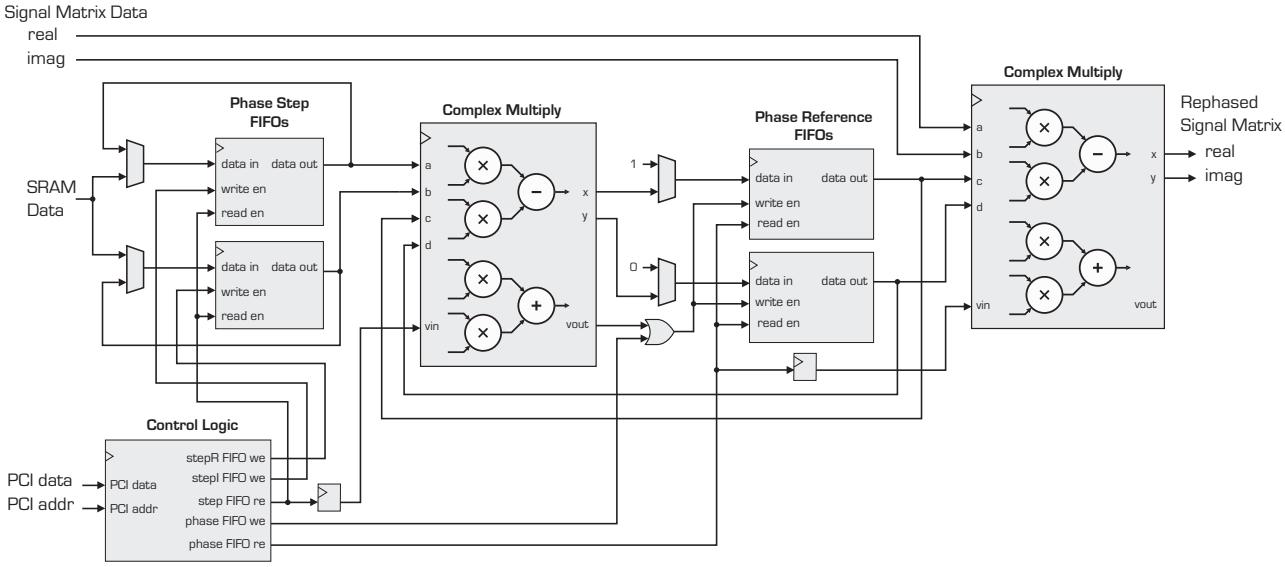

Figure 4.15: Rephasing stage decompression and application circuit

4.9 QR Decomposition Stage

The QR decomposition stage reduces the rephased signal matrix to upper triangular form. This operation has the important property of reducing the number of elements in the input matrix, without altering its singular values (see Sections 2.5.1 through 2.5.3). This introduces a form of compression for narrow rectangular matrices, which is useful in the context of FPGA-based processing because it reduces the amount of data that must be stored on-chip, where memory resources are limited. For example, when processing a signal matrix with dimensions of 128-by-16, the result is a 16-by-16 upper triangular matrix, and the

compression ratio is better than 1:8. In order to retain this benifit of QR decomposition in a parallel implementation, the operation must be implemented by exploiting low level parallelism. That is, it is not good enough to simply run multiple, independent, sequential-type QR decomposition modules in parallel, because each one will require a full-size input buffer or some other large data memory.

In order to exploit the data compression afforded by QR decomposition, the operation was implemented using a linear array of processors. Data enters the array in bursts of one matrix row at a time, and exits the array at the same rate. Each element in the processing array requires storage space for only three matrix rows. Each array element eliminates the subdiagonal matrix elements in one column of the input matrix, so n processors are required for processing an m-by-n signal matrix. The matrix that exits the array is upper triangular, and requires a smaller buffer than would be needed to store the rectangular input matrix.

The linear array approach also results in a scalable system. The length of the processing array may be increased for wider matrices. Widths of up to 265 are easily supported given on-chip memory constraints. The height of the signal matrix is limited only by the fact that sub-diagonal energy in the matrix will be concentrated onto the diagonal elements, and will eventually cause overflow of the fixed-point data representation. Due to DSP and CLB constraints, a single SX55 FPGA is capable of supporting up to 31 processing elements. For matrices with more then 31 columns, multiple FPGAs may be used. Because the processing array is linear, and not two-dimensional, the IO bandwidth requirement between devices is achievable. Only two 48-bit buses, operated at the processing clock rate, are required to interconnect one FPGA to both of its neighbors.

Because the number of processing elements grows as the width of the signal matrix is increased, the processing performance of the array also scales up. This means that, when compared to sequential processing machine, the parallel QR implementation will provide greater speed-ups for larger matrices than for smaller matrices. In essence, the processing time required for QR decomposition of an m-by-n matrix is reduce from $O(mn^2)$ to $O(mn)$.

4.9.1 Algorithm

Recalling the Givens Rotation method for performing QR decomposition in Section 2.5.3, the algorithm dictates that all sub-diagonal elements in one column should be annihilated before proceeding to the next column. Viewed from a different perspective, i.e. that of a parallel system specification, the requirement states: for a processing element that eliminates sub-diagonal matrix elements in column k, all sub-diagonal matrix elements in column $k-1$ that enter this processing element must be zero. This statement is essentially self-fulfilling assuming that the first processing element eliminates sub-diagonal matrix elements in the first column, the second processing element eliminates matrix elements in the second column, et cetera. The only challenge is to determine how eliminate each sub-diagonal element shortly after it arrives, without knowledge of distant row content, so that only a few rows need to be stored in each processing element.

The sub-diagonal element annihilation task can be localized to two adjacent rows if the elimination processing begins at the bottom of the matrix, row m, and if the k^{th} Givens rotation operation is performed such that it involves moving energy from row $m-(k-1)$ to row $m+k$. This is exactly the method employed in the QR decomposition processing array. Each new row that arrives is rephased such that phase of its leftmost non-zero element (the target element) is set to zero as described by Equations 4.6, 4.9. and 4.10. The magnitude of this element is then compared to the magnitude the corresponding element in a "feedback" row, as described by the arctangent operation in Equation 4.3. The Givens rotation described by Equations 4.13 and 4.14 is then performed in order to rotate the energy of the target element into the feedback row. The row whose target element has been annihilated is then passed to the next processing element. When the final target element (which lies on the matrix diagonal) is reached, the last Givens rotation is performed and the feedback result is deposited on the diagonal. Rows above the final row are part of the upper-triangular result and need not be processed.

Because one column is eliminated by each processing element, the number of non-zero elements that must be processed by subsequent processing elements is reduced. That means that some processing elements have a lower processing load than other processing elements. This leads to less efficient use of resources in processing elements located towards the end of the array. In order to minimize this effect, processing elements with complementary sized

loads share a vector processing unit (see Section 4.5). For example, if the width of the matrix is 16, then the first processor may be considered fully loaded, with 16 non-zeros elements. The second processor, however, is only loaded with 15 non-zero elements. The last processor is loaded very lightly, with 1 non-zeros element. Loads of 15 and 1 are complementary because they sum to 16. Therefore, processing elements 2 and 16 share a VPU. Elements 3 and 15 have complementary loads of 14 and 2, and also share a VPU. The same is true for processing elements 3 and 14, 4 and 13, 5 and 12, et cetera. If the processing elements are arranged properly with in the FPGA then the shared resources may be kept close to other logic within the processing element, keeping signal travel distance short. The appropriate arrangement is a U shape, as shown in Figure 4.16 [28].

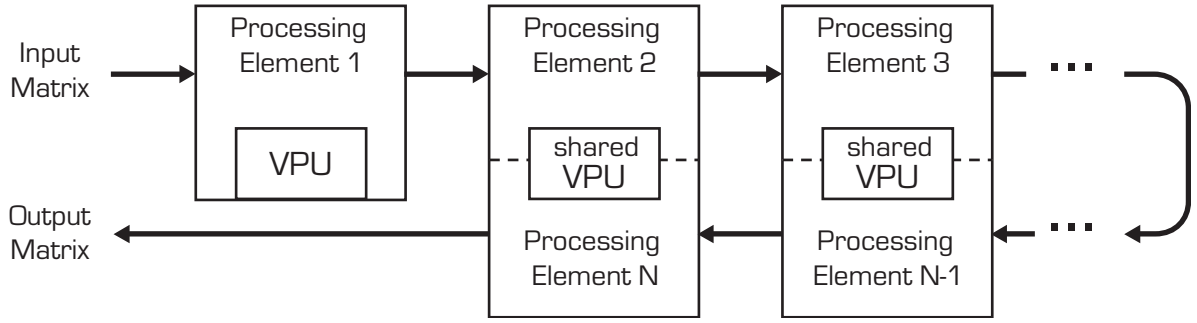

Figure 4.16: Elements in the QR decomposition processing array share vector processing unit (VPU) resources in order to improve efficiency. The elements are arranged in a U shape so that elements that share a VPU remain close to each other. This helps to avoid long-distance signal routing.

4.9.2 Processing Element Architecture

Further parallelization, in addition to that afforded by the array approach, is achieved using a coarse row-by-row pipeline. That is, the phase angle of the target element in row k is computed while the rephasing and the arctangent computation for row $k+1$ are performed and while the Givens rotation of row $k+2$ is computed. Buffering of the next input row, $k-1$, also occurs in parallel with these operations. The architecture of the QR decomposition processing element is divided into stages that roughly reflect this parallelization. The four stages, which are described in more detail in following sections, are the

- Receive stage

- Measure and compare stage (CORDIC stage)
  These stages have been grouped because they share CORDIC resources
- Processing stage (VPU stage)
- Output stage

Because two processing elements share a single VPU (as discussed in the previous section), complementary elements were grouped into a single processing element design which has two inputs, and two outputs for connecting to its neighbors. A top-level diagram of this architecture is shown in Figure 4.17. Data from neighboring processing elements are buffered in the receive stage. Once a full row has been collected processing begins. The buffered rows are read from the receive buffers inside the receive stage, and the non-zero elements from each row are written to the primary buffer. The target elements from each row are observed by the measure portion of the measure and compare stage as they are passed to the buffer. For each target element, the measure stage computes a complex-conjugate unit vector that will be used to rephase the rows such that the target elements have phase angles of zero. The measure stage also computes the magnitudes of the target elements, which are passed to the compare stage. The processing stage reads the new row data from the primary buffer, rephases the row elements using the vector from the measure stage, and stores the results in its internal buffers. The compare stage computes the arctangent function indirectly, by generating unit vectors for each row as described by Equation 4.8. These unit vectors are used within the processing stage to perform the Givens rotations that annihilate the target elements, and produce feedback rows. The rows whose target elements have been eliminated are passed to the next processing element; the feedback rows are stored in a buffer within the processing stage. For rows that lie above the matrix diagonal, no processing is performed, so these rows must be passed directly to the next stage. This is accomplished by the output stage, which reads these rows from the receive stage, and buffers them so that they may be pass along at the appropriate time (i.e. after the pipeline delay). A processing schedule for the QR decomposition processing array element is shown in Figure 4.18

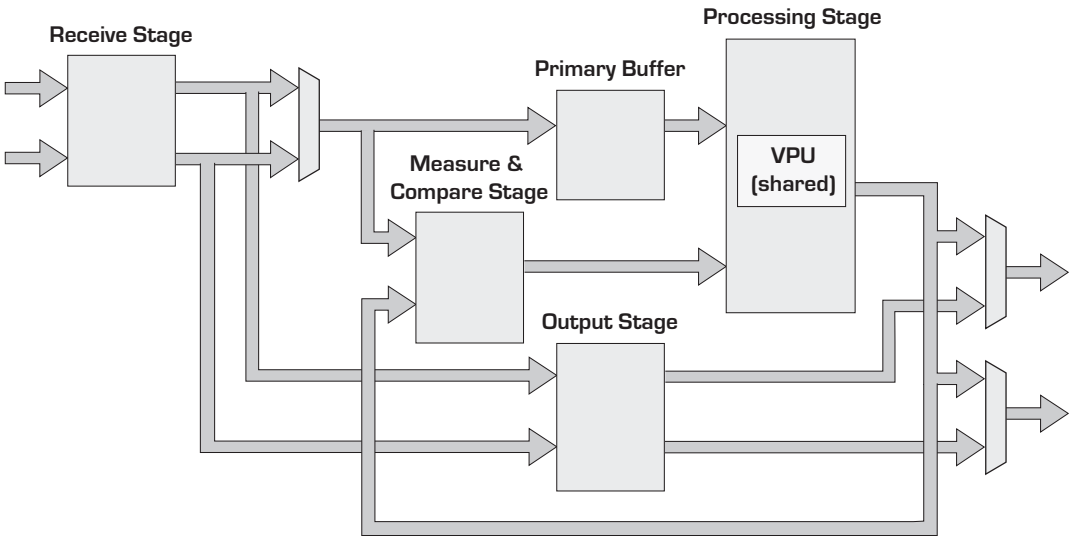

Figure 4.17: Top-level diagram of an element from the QR decomposition processing array. Two processing elements, which share a vector processing unit, have been grouped into a single processing element design with two inputs, and two outputs for connecting to its neighbors.

	Receive Stage	Measure and Compare Stage		Processing Stage	Output Stage
		Measure	Compare		
t = 1	buffer row K	compute unit vector for row K+1 rephasing	compute unit vector for row K+2 Givens	rephase row K+2 Givens of row K+3	output row K+3 if above diagonal
t = 2	buffer row K-1	compute unit vector for row K rephasing	compute unit vector for row K+1 Givens	rephase row K+1 Givens of row K+2	output row K+2 if above diagonal
t = 3	buffer row K-2	compute unit vector for row K-1 rephasing	compute unit vector for row K Givens	rephase row K Givens of row K+1	output row K+1 if above diagonal
⋮	⋮	⋮	⋮	⋮	⋮

Figure 4.18: QR decomposition processing element: processing schedule

4.9.3 Receive stage

The receive stage is responsible for buffering row data that arrive from neighboring processing elements. The receive stage also counts each incoming row and, using information about the size of the matrix and the target column, classifies each row by type. Subsequent stages in the QR decomposition processing element use this type information to determine what kind of processing is required for each row. The row type classifications are: sub-diagonal, diagonal, and super-diagonal. For a processing element that seeks to annihilate subdiagonal elements in column k, the classification for row j is

$$rowtype = \begin{cases} subdiagonal & : \quad j > k \\ diagonal & : \quad j = k \\ superdiagonal & : \quad j < k \end{cases} \tag{4.24}$$

A diagram of the receive stage is shown in Figure 4.19. Row data arrive from neighboring processing elements via ports A and B along with corresponding "valid input" signals. The data are counted and collected in FIFO buffers. When an entire row is received from neighbor A or B, a corresponding flip-flop is set. Arrival of a new row also causes the row counter to be decremented. The row counts are decoded to obtain the aforementioned row type classification. If no new row is available, the row type is set to "invalid". This designation is necessary in order to alert subsequent stages in the case that there is a row available from one neighbor, but not the other. The occurs when the processing pipe is first being filled. The processing of a received row is triggered when the "go" signal is asserted. If a new row is available from A, B, or both A and B, then the row types are latched, and the new row signal is asserted for one cycle. Subsequent stages assert the read enable signals to retrieve the row data.

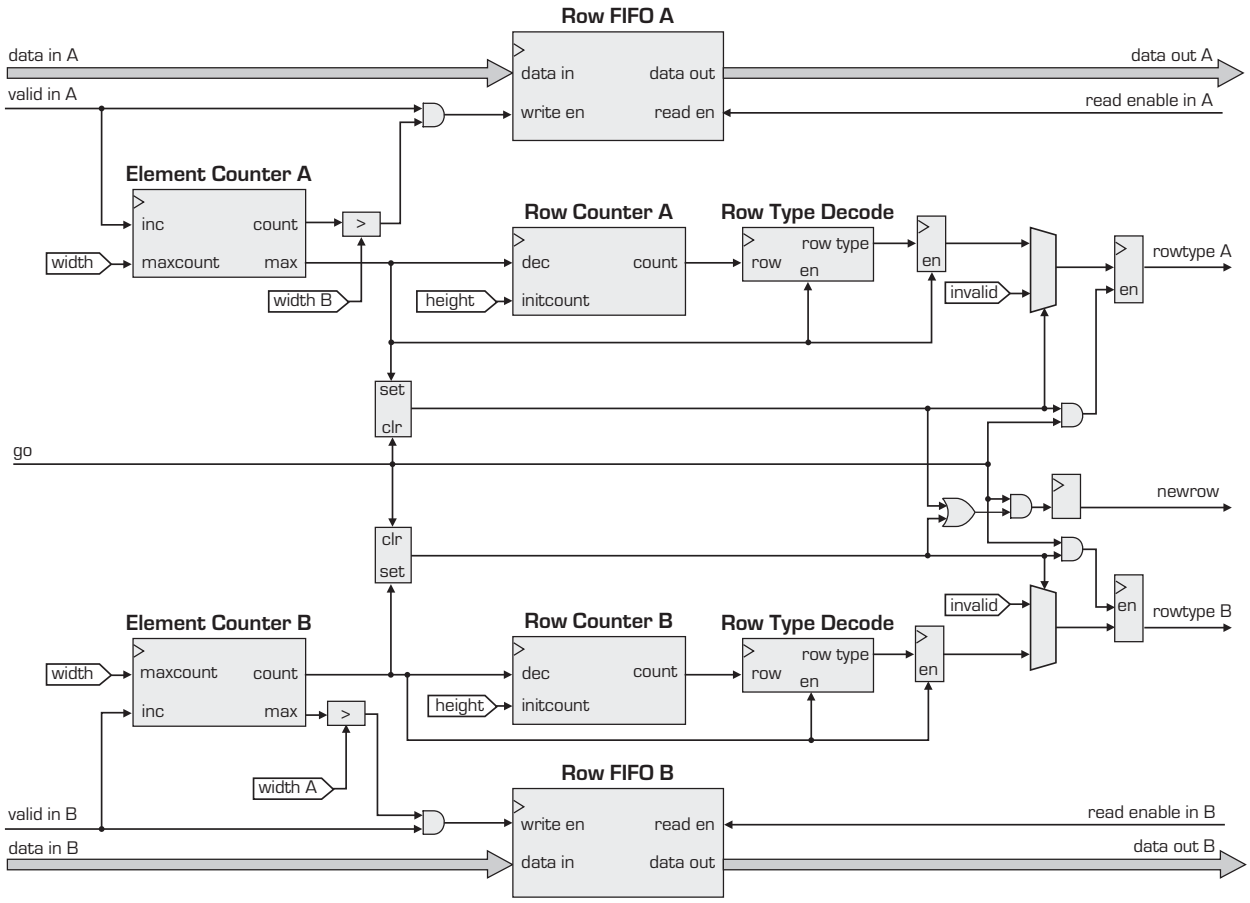

Figure 4.19: Receive stage of the QR decomposition processing element

4.9.4 Measure and Compare stage (CORDIC stage)

The measure and compare stage employs the CORDIC module, described in Section 4.6, to compute unit vectors and vector magnitudes. Two CORDIC modules are used; one processes data from processing element neighbor A, while the other processes data from neighbor B. A diagram of the measure compare stage is shown in Figure 4.20.

For each new subdiagonal or diagonal row that arrives, the measure and compare stage produces a read enable signal that causes the row data to be retrieved from the receive stage, and transferred to the primary buffer. The values of the A and B target elements are registered as they are transfered, so that they may be used to generate a unit vectors for reducing their phase angles to zero (see Eq. 4.6) . The magnitudes of the target elements are also computed by the CORDIC module, and registered. These magnitudes are used in the calculation of unit vectors that will be used during the Givens rotations involving the new rows (see Eq. 4.8) . The other values required for the calculation of the Givens rotation unit vectors are generated by the processing stage and arrive at the measure-compare stage via the feedback A and B inputs. The arrival of these values, which are equal to the magnitude of the target element in the current feedback row (see Section 4.9.1), are indicated by the assertion of the "mark A" and "mark B" signals generated by the processing stage.

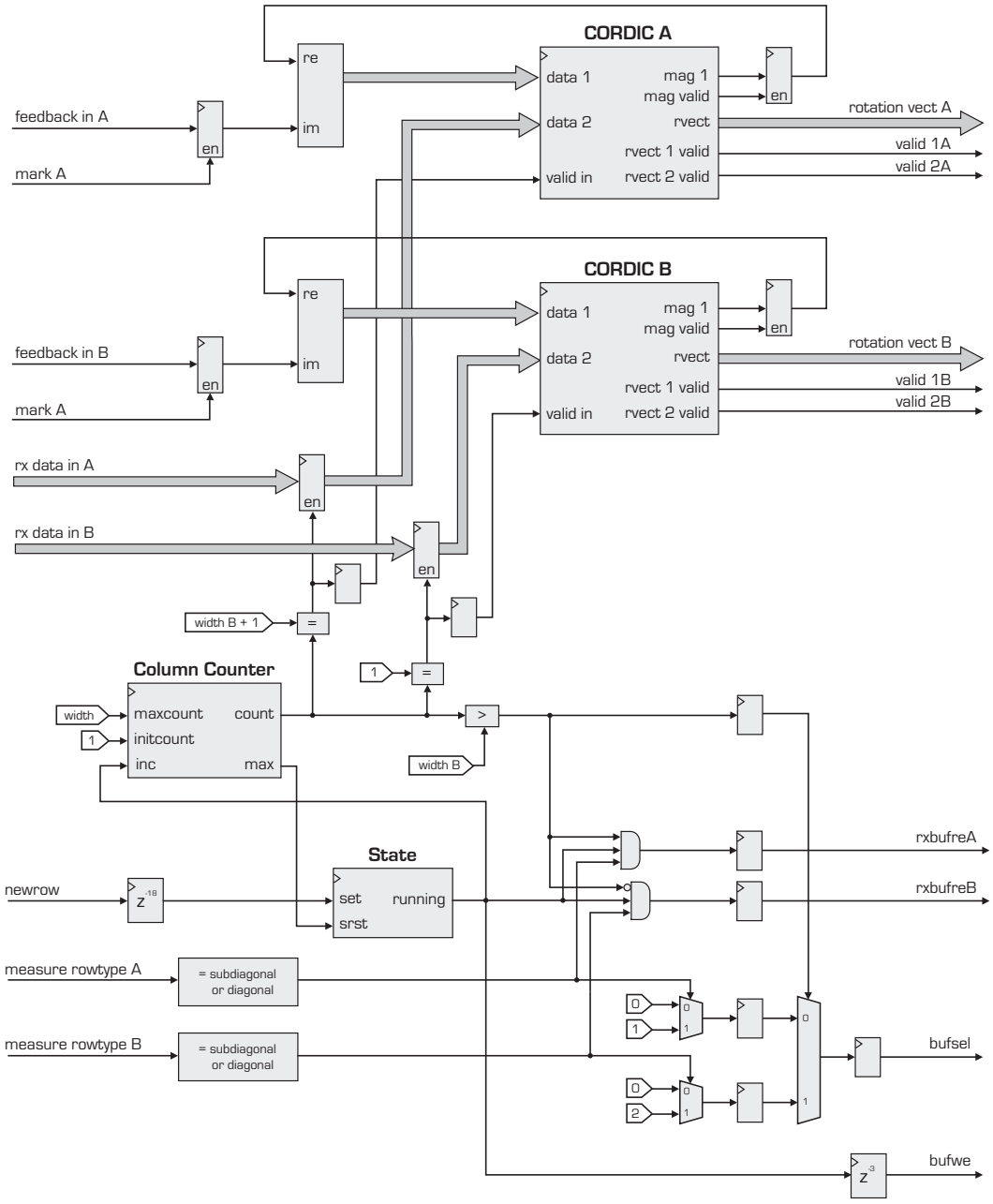

Figure 4.20: Measure and compare stage of the QR decomposition processing element

4.9.5 Processing stage (VPU stage)

The processing stage is responsible for performing the vector operations described in Equations 4.9 through 4.16. Because the phase of the target element in the feedback row is guaranteed to be zero by the chosen QR decomposition algorithm (see Section 4.9.1), only three of these operation pairs need to be computed for each new row. These operations have been designated as

- Rotation:

$$\Re\{\vec{a_j}'\} = \Re\{u_j\}\Re\{\vec{a_j}\} - \Im\{u_k\}\Im\{\vec{a_j}\} \tag{4.25}$$
$$\Im\{\vec{a_j}'\} = \Re\{u_j\}\Im\{\vec{a_j}\} + \Im\{u_k\}\Re\{\vec{a_j}\} \tag{4.26}$$

- Output:

$$\Re\{\vec{a_j}''\} = \Re\{u_0\}\Re\{\vec{a_j}'\} + \Im\{u_0\}\Re\{\vec{a_k}'\} \tag{4.27}$$
$$\Im\{\vec{a_j}''\} = \Re\{u_0\}\Im\{\vec{a_j}'\} + \Im\{u_0\}\Im\{\vec{a_k}'\} \tag{4.28}$$

- Feedback:

$$\Re\{\vec{a_k}''\} = \Re\{u_0\}\Re\{\vec{a_k}'\} - \Im\{u_0\}\Re\{\vec{a_j}'\} \tag{4.29}$$
$$\Im\{\vec{a_k}''\} = \Re\{u_0\}\Im\{\vec{a_k}'\} - \Im\{u_0\}\Im\{\vec{a_j}'\} \tag{4.30}$$

Note that the output and feedback operations together comprise a Givens rotation.

Within the processing stage, the results of these operations are computed sequentially. That is, to process a row with n elements, $3n$ clock cycles are required. This splitting of the Givens rotation into two parts results is a better balance with the processing requirements of the phase rotation operation, which requires half as many computations as the Givens rotation. The structure of the processing stage is shown in Figure 4.21.

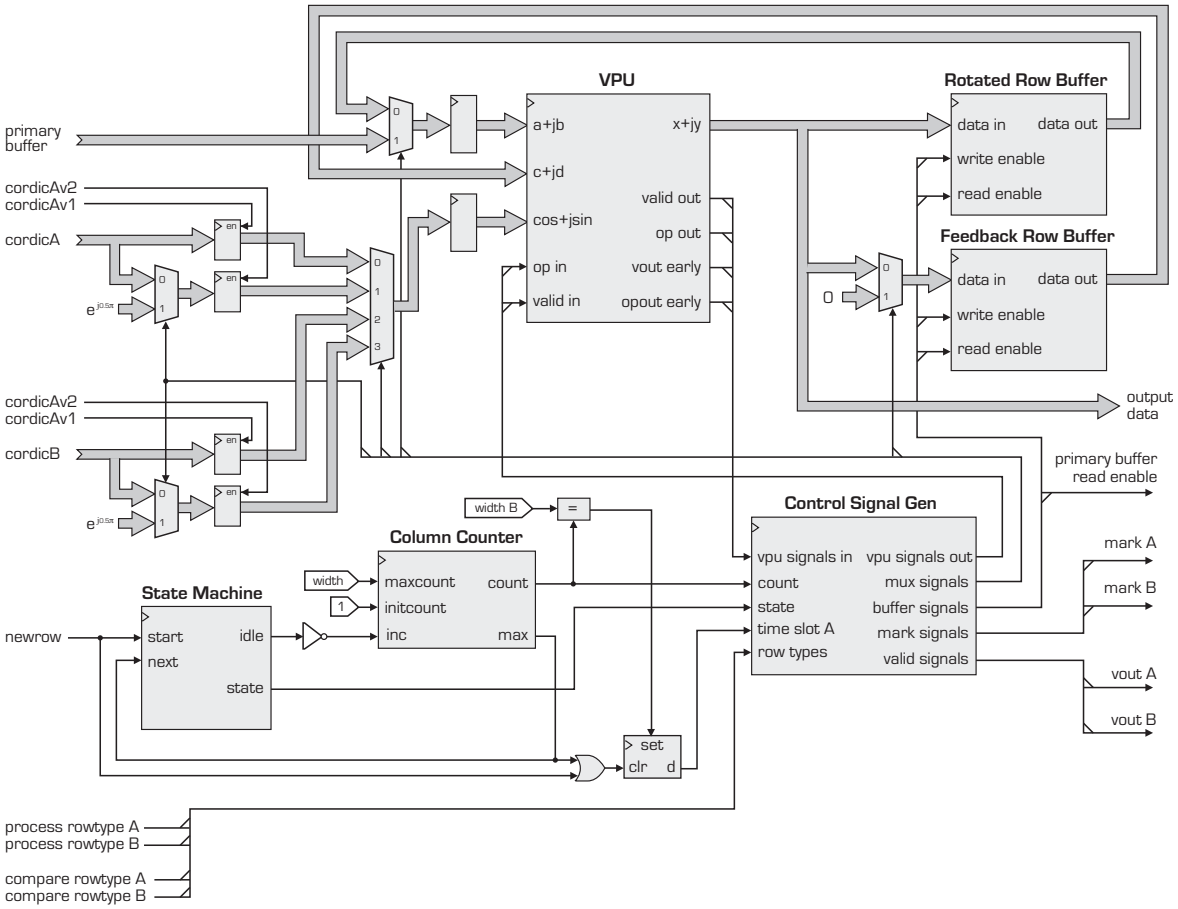

Figure 4.21: Processing stage of the QR decomposition processing element

Computations within the processing stage are triggered by the assertion of the "newrow" signal, which causes the state machine to move from the idle state, to the feedback stage. For every cycle that the state is not equal to the idle state, the column counter is incremented. When the count reaches the number of columns in the signal matrix, *width*, the counter rolls over, and the state is updated according to the state-flow diagram in Figure 4.22.

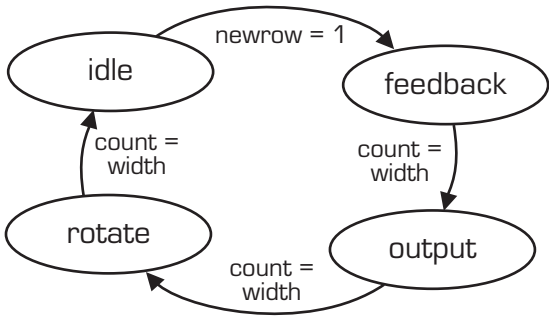

Figure 4.22: State-flow diagram for the processing stage

In order to achieve the resource sharing discussed in 4.9.1, each of the processing states is divided into two time slots, A and B, during which data from the two neighboring processing elements are processed. This is illustrated in Figure 4.23.

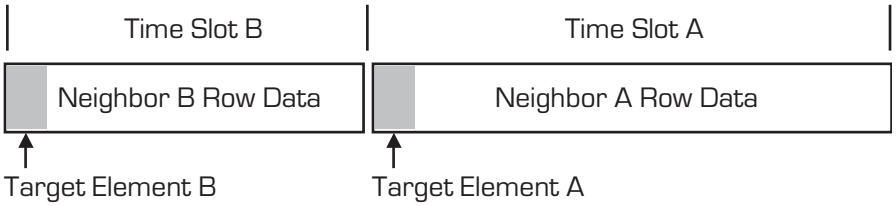

Figure 4.23: Resource sharing schedule for the processing stage. Row data from neighboring processing elements, A and B, present complementary loads, which are combined to acheive better load balancing across all processing elements.

Feedback State: When the processing stage is in the feedback state, partial Givens rotation operations are performed using the row data contained in the "rotated row" and "feedback row" buffers connected to the $a + jb$ and $c + jd$ input ports of the VPU. The $cos + jsin$ port is connected to the Givens rotation unit vectors, which have been generated by the measure and compare stage, and registered internally. For time slot A, the result produced by CORDIC module A is used. For time slot B, the result from CORDIC module

B is used. The VPU is set to compute Equations 4.29 and 4.30, and the results are queued in the feedback buffer. As the A and B target elements are written to the feedback buffer, the corresponding "mark A" and "mark B" signals are asserted. This allows the measure and compare stage to register these results, so that they may be used for calculating the next Givens rotation unit vector (described by Equation 4.8). When a diagonal row for A or B has processed (see Section 4.9.3) the $cos + jsin$ port is connected to a unit vector with phase angle of 90 degrees during the appropriate time slot. This causes the feedback row to be deposited on the matrix diagonal as described in Section 4.9.1.

Output State: The output state is very similar to the feedback state. The same unit vectors are used, and the remaining portion of the Givens rotation is computed. The only change is the operation of the VPU, which is set to compute Equations 4.27 and 4.28. The target elements of the resulting A and B rows are zero. These rows are passed directly to subsequent elements in the QR decomposition processing array.

Rotate State: When the processing stage is in the rotate state, the newly arrived row data are rephased such that the A and B target elements have phase angles of zero. This is accomplished by selecting the unit vectors generated by the A and B CORDIC module in the measure stage. The rephased data is stored in the "rotated row" buffer as it exits the VPU, and will be used for the next Givens rotation.

4.9.6 Output Stage

Rows that have been classified as superdiagonal by the receive stage (see Section 4.9.3) do not need to be processed, because they are part of the upper-triangular output matrix. The purpose of the output stage is to read these rows out of the receive stage buffers, and store them until the time is appropriate for them to be sent to the next processing element in the QR decomposition processing array. In order to match the delay of the processing pipe, the buffered data are passed along three time-periods after they are received as shown in the processing schedule (Figure 4.18). The structure of the output stage is shown in Figure 4.24.

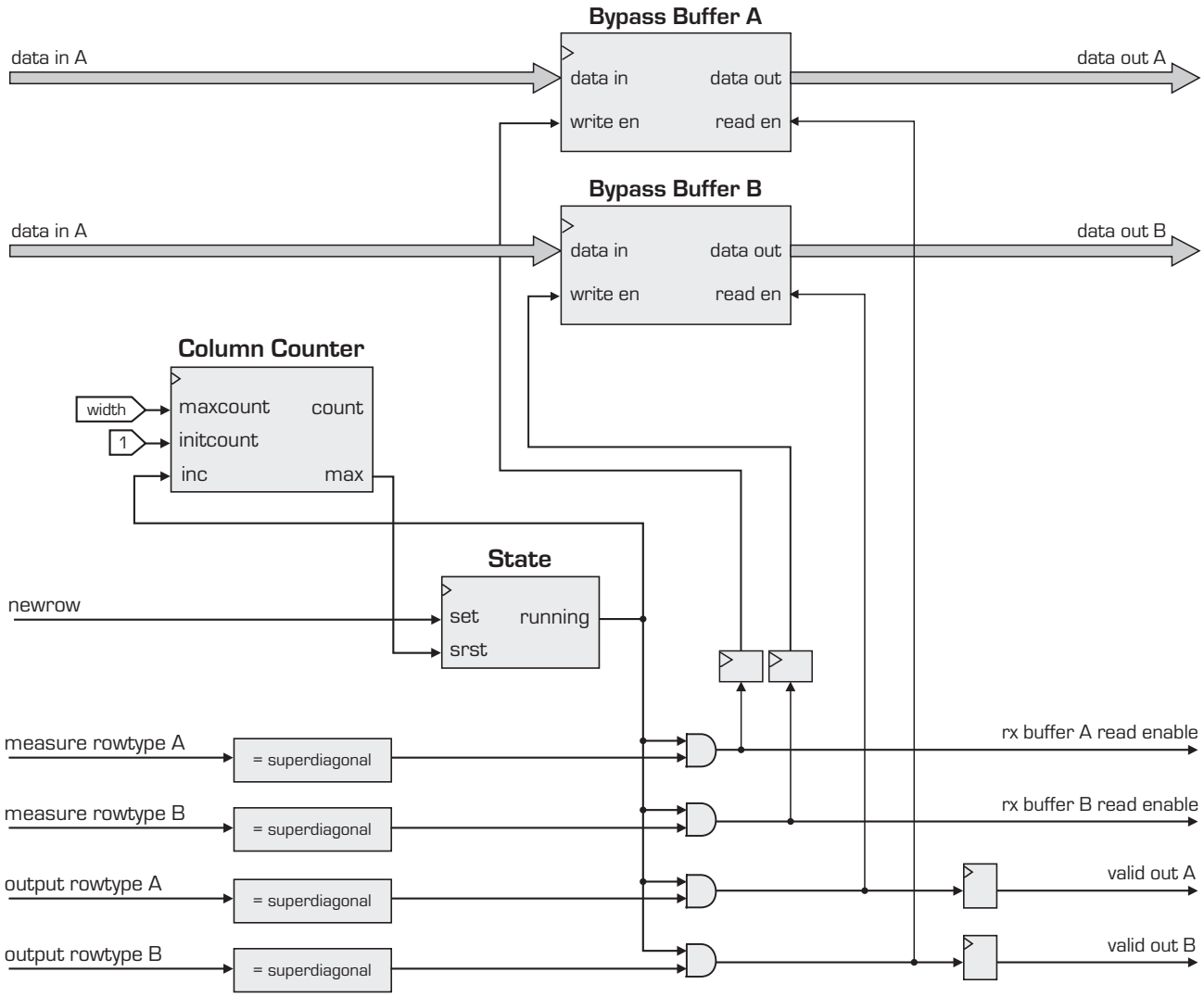

Figure 4.24: Output stage of the QR decomposition processing element

The output stage becomes active when a new row arrives, as indicated by the assertion of the new row signal. The row type labels corresponding to the newest row data are used to determine if the available rows are superdiagonal. If either the A or B row data correspond to a super-diagonal row, then they are read from the receive stage buffers, and written to the appropriate "bypass" buffers within the output stage. The row data remain in these buffers for three time-periods, until the output row type becomes super-diagonal. At this time, the row data are read from the bypass buffers, and presented on the data output ports of the output stage, along with corresponding valid signals.

4.10 Bidiagonalization Stage

Matrices that have been converted to upper-triangular form by the QR decomposition processing array are passed to the bidiagonalization stage. The purposed of the bidiagonalization stage is to annihilate all but the diagonal and super-diagonal elements of these matrices before they are passed to the host PC. The chosen bidiagonalization algorithm is similar to the QR decomposition algorithm discussed in Section 4.9.1 in the sense that the same feedback accumulation method is employed. In the bidiagonalization stage, however, this technique is not employed in order to facilitate linear array processing. As discussed in Section 2.5.5, bidiagonalization involves annihilation operations that alternate between the elimination of sub-diagonal column elements and the annihilation of row elements to the right of the matrix superdiagonal. These operations will be referred to as row and column reductions. The ordering of the reduction process is illustrated in Figure 4.25. This necessary ordering results in a data dependency between subsequent annihilation steps. For example, reduction of row 1 to bidiagonal form involves altering columns 2 and greater. Therefore, reduction of column 2 must be completed after the reduction of row 1 is complete.

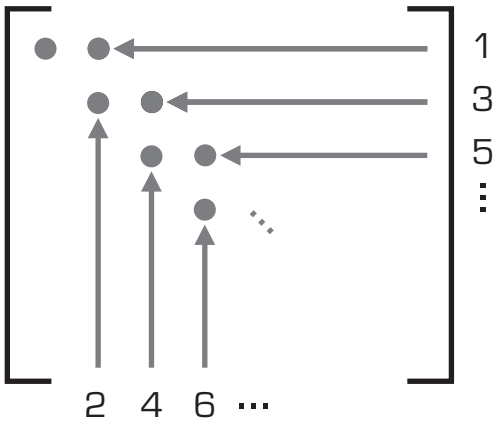

Figure 4.25: The bidiagonalization process alternates between column element and row element annihilation (as indicated by the numbers below and to the right of the matrix shown here). Because each row annihilation alters matrix elements that will be used in subsequent column annihilations (and visa versa), there is a data dependency between subsequent annihilation steps.

Because of this data dependency, the bidiagonalization operation was implemented within a single processing element, instead of using a linear processing array. In order to match

the processing performance of the QR decomposition stage, the bidiagonalization stage is implemented as a group of bidiagonalization modules, which operate independently, but in parallel. The current implementation of the SART co-processor system uses four bidiagonalization modules. For signal matrices of different sizes, more or fewer modules may be needed in order to match the throughput of the QR decomposition stage.

Further data dependencies exist within the bidiagonalization algorithm. Namely, the computation of the unit vectors described in Equations 4.6 and 4.8 must be completed before the rotation operations described by Equations 4.11 through 4.16 may be performed. In order to overcome this dependency, and achieve efficient use of logic resources, each bidiagonalization stage processes two matrices in an interleaved fashion. Unit vector computations for one matrix are performed while rephasing and Givens rotation operations are computed for the other matrix, as indicated in the processing schedule in Figure 4.26.

	CORDIC Module Input		Vector Processing Unit Input	
	Phase rotation unit vector computation	Givens rotation unit vector computation	Phase rotation operation	Givens rotation operations
t = 1	Matrix 1 Row K	–	–	–
t = 2	Matrix 2 Row K	–	Matrix 1 Row K	–
t = 3	Matrix 1 Row K-1	Matrix 1 Row K	Matrix 2 Row K	–
t = 4	Matrix 2 Row K-1	Matrix 2 Row K	Matrix 1 Row K-1	Matrix 1 Row K
t = 5	Matrix 1 Row K-2	Matrix 1 Row K-1	Matrix 2 Row K-1	Matrix 2 Row K
t = 6	Matrix 2 Row K-2	Matrix 2 Row K-1	Matrix 1 Row K-2	Matrix 1 Row K-1
⋮	⋮	⋮	⋮	⋮

Figure 4.26: Processing schedule for bidiagonalization module. The highlighted boxes show the data dependency between operations. By interleaving the bidiagonalization of two matrices no resources are left idle (apart from a three time-period interval, during which the processing pipeline is filled).

4.10.1 Top-Level

As mentioned in the previous section, the bidiagonalization stage of the SART co-processor system is implemented as a group of processing modules that operate independently and in parallel. As shown in Figure 4.27 each module contains a dual port data memory that is connected to the output of the QR decomposition stage. Upper-triangular matrices generated by the QR decomposition stage are written to these memories in a "round-robbin" fashion, as orchestrated by a simple scheduling module. Each data memory is large enough to store four matrices. At any given time, two of these matrices are being processed using the interleaved technique discussed in the previous section. The remainder of the data memory is reserved so that two additional matrices may be loaded while the current matrices are being processed. At their output side, each bidiagonalization module contains a result buffer. The diagonal and super-diagonal elements produced during the bidiagonalization of 32 matrices can be stored in each result buffers. This allows the host system to trigger a burst of SART calculations, perform other calculations locally while the results are gathered, and then retrieve the results from the SART co-processor when the are ready.

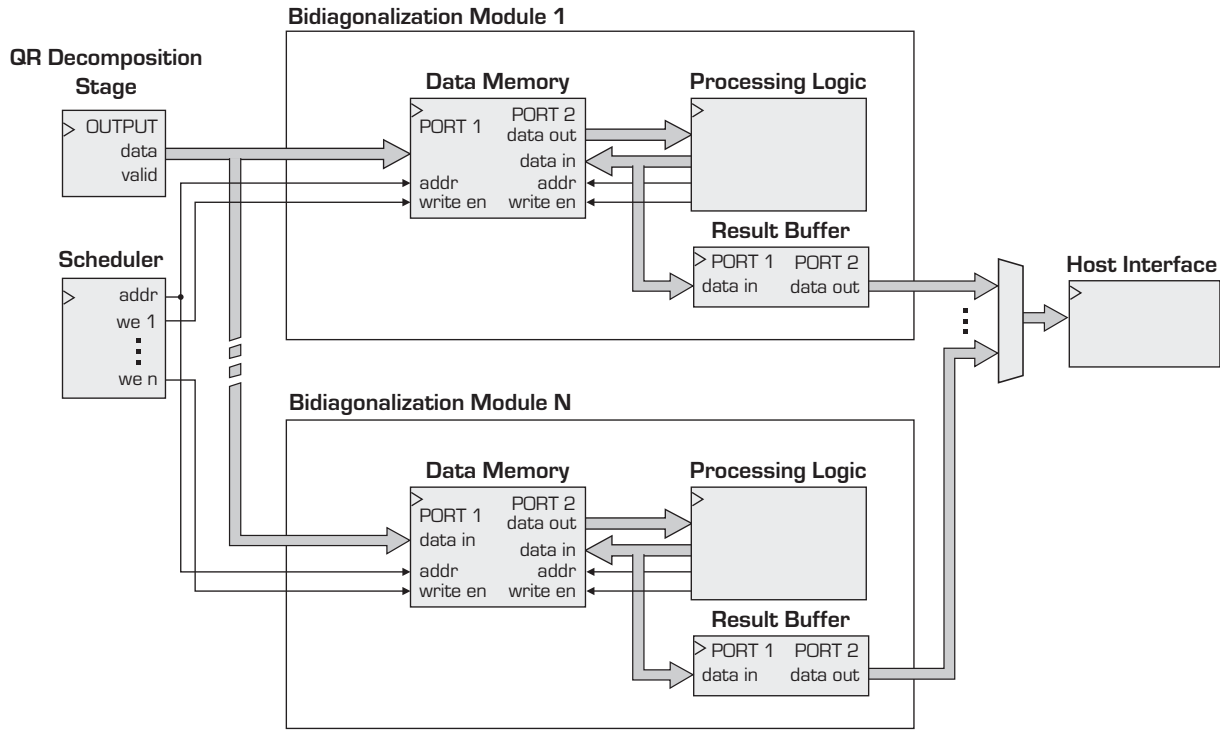

Figure 4.27: Bidiagonalization stage of the SART co-processor system

4.10.2 Bidiagonalization Module Architecture

The top-level structure of a single bidiagonalization module is shown in Figure 4.28. Upper-triangular matrices from the QR decomposition are loaded into main memory via the input data bus, which is controlled by the scheduling module. Unlike the QR decomposition processing element, the bidiagonalization module holds matrix data within this local memory during processing because each matrix element must be accessed many times during processing. After a set of two matrices have been loaded, the "program start" signal is asserted and processing begins.

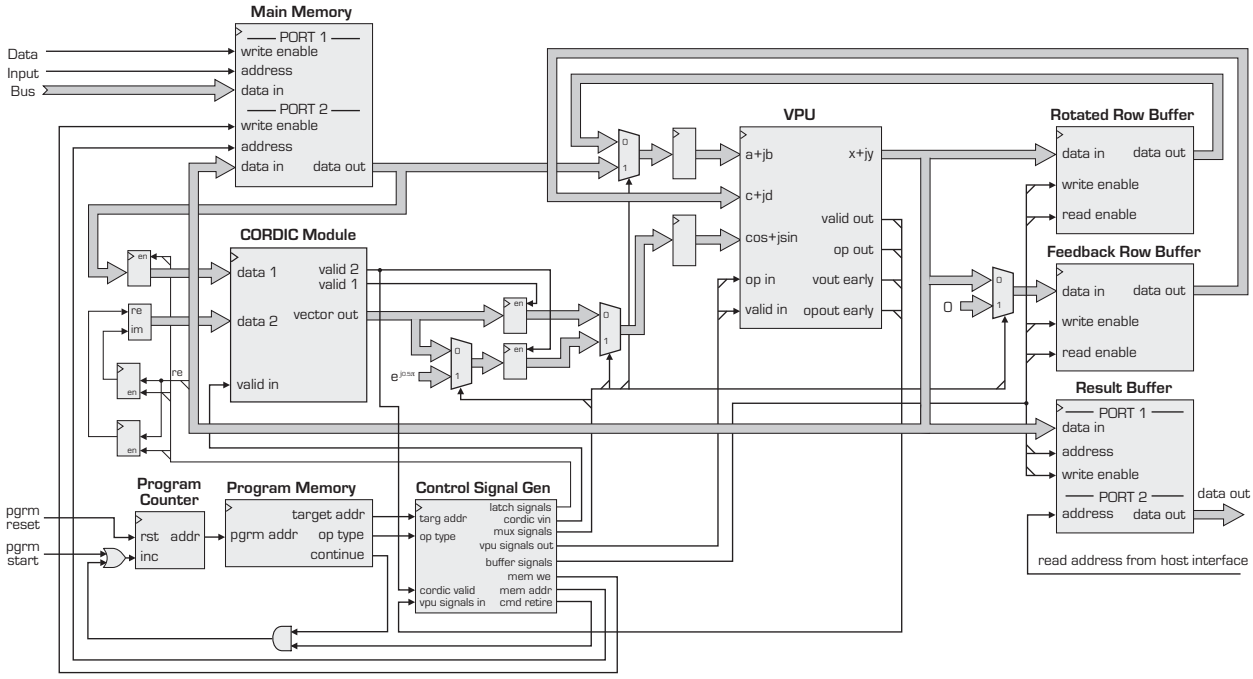

Figure 4.28: Bidiagonalization module structure

Just as in the QR decomposition module, target rows and columns are rotated such that their target elements have a phase angles of zero. Rephased rows and columns are stored in a buffer that is separate from main memory. This approach is quite benificial, because it essentially doubles the bandwidth of the working memory. Despite the fact that one port of the dual-port main memory is reserved for the input interface, two pieces of matrix data are accessed on each clock cycle. The same is true for the other processing buffer, which holds the "feedback" row or column that accumulates the energy from each target element that is

annihilated. Processing involves the same three operations that were employed in the QR decomposition processing stage (Section 4.9.5), namely: rotation, output, and feedback. In this case, however, the output does not involve output to a neighboring processing element, but rather a write-back to main memory. Figure 4.29 shows how data flow between main memory and the intermediate result buffers during each operation.

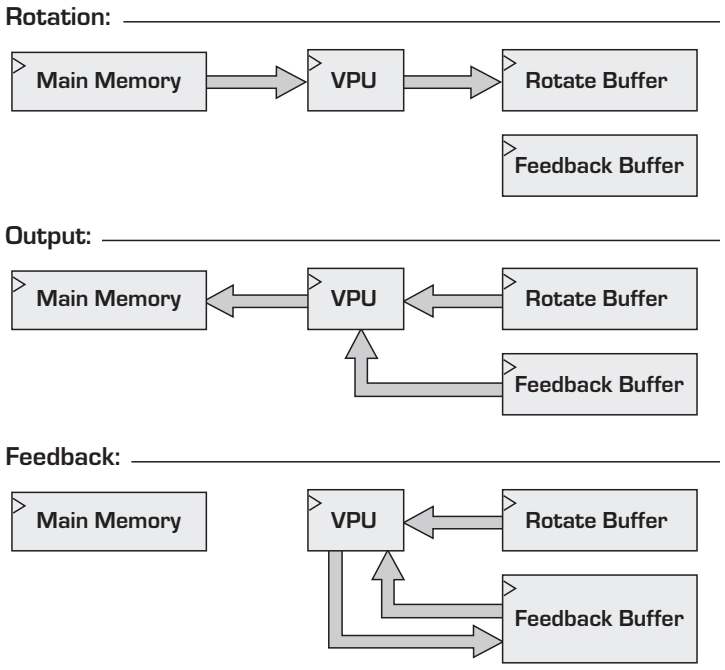

Figure 4.29: Data flow diagram for the bidiagonalization module showing how matrix data flow between main memory and the intermediate result buffers during each operation type

4.10.3 Processing Program and Command Sequencing Macro

Control of the bidiagonalization module was implemented using a programmatic approach. By employing a program sequence that can be altered, the verification and debugging process was better facilitated. For example, the program could be quickly adjusted such that the module performed one element annihilation, one row or column reduction, or the full bidiagonalization process.

The format of the bidiagonalization program is very simple. A command specifies which matrix element should be targeted for anhihilation next, and how the annihilation should

occur. One row or column reduction sequence is encoded in multiple commands; the number of commands is equal to the number of matrix elements which must be annhilated. Each command contains three pieces of infomation:

- The memory address of the element targeted for annihilation
- The type of Givens rotation (pre- or post-multiplication)
- An end of program indication

Each command that is issued effects the operation of the bidiagonalization module for many cycles as it is expanded according to the command sequencing macro. As suggested by the processing schedule from Figure 4.26, the processing pipeline has a depth of four commands. When a command is issued, the specified target element is read from memory and passed to the CORDIC module for phase angle calculation. This is the first processing phase. During the next time period, a new command is issued, unless the end-of-program indicator has been reached, in which case the program terminates. In either case, the previous command progresses to the next stage in the processing pipeline.

The second processing phase involves rotating the target column or row so that the target element phase becomes zero. If the command specified a Givens rotation by pre-multiplication, then rephasing effects a matrix row. If the command specified post-multiplication, then a column is effected. The rephased row or column is stored in the rotation buffer.

The third processing phase involves calculating the unit vector that will be used during the Givens rotation operation that will annihilate the target element. The inputs to the CORDIC module for this calculation are the magnitude of the target element, which is simply the real component of the target element after rephasing, and the magnitude of the target element in the feedback buffer, which starts at zero and increases as a column or row reduction proceeds. These values are latched in a pair of registers as they appear at the output of the VPU.

During the fourth phase of processing in the bidiagonalization module pipeline, the Givens rotation is performed. This computation is divided into two halves, just as in the QR decomposition processing stage. During any given processing period, the VPU performs a sequence involving each of its three operations: rephasing of the next target row or column (rotation), half of a Givens rotation that annihilates the target element in the current row,

and then the second half of the Givens rotation, which produces the feedback result. If the operation is the last in a series of operations that implement one row or column reduction, then the feedback result is written to main memory, and the diagonal and subdiagonal elements are also written to the result buffer. Each result buffer contains two banks. One bank is used to collect bidiagonalization results; simultaneously, host system may retrieve the results of previous computations from the second bank.

As depicted in Figure 4.26, a command that has been expanded according to the sequencing macro occupies either the CORDIC module or the VPU during the four processing periods for which it is active. In fact, a command alternates between these resources, and never occupies both. Because idle resources result in an inefficient design, two matrices are processed in an interleaved fashion. That is, when elements from matrix 1 are being processed by the CORDIC module, elements from matrix 2 are being processed by the VPU. This approach increases the efficiency of the design, however, the processing pipeline must be flushed after each reduction sequence due to data dependency. For example, the last command in reduction operation that annihilates subdiagonal elements in column two of the matrix will involve alteration of row two of the matrix. Therefore, the next reduction operation, which will eliminate elements to the right of the super-diagonal in row two, may not begin until the column two reduction operation is complete.

After the final command in the processing program has been completed, the signal matrix is in bidiagonal form. The non-zero matrix elements are contained in the output buffer, and may be retrieved by the host system, and diagonalized in order to obtain the singular values of the signal matrix. Operation of the SART co-processor system, including the final diagonalization step, is described in Chapter 5.

Chapter 5

Operation

This chapter contains information relating to the operation of the SART co-processor system. The first two sections describe how data are loaded into the co-processor. This includes loading the scan-grid phase reference matrix into the system's SRAM, and loading signal matrices into the FPGA for each SART scan. The third and fourth sections describe how SART calculations are triggered, and how the results of these computations may be retrieved. For reference, tables containing the addresses for all memory banks, and control signals with the SART co-processor system are also provided. The final section in this chapter describes how the final SART processing stage, diagonalization, may be performed in software once the bidiagonal signal matrices have been retrieved from the co-processor result buffer.

5.1 Loading the Scan-Grid

The first step in operating the SART co-processor system involves loading the SART scan-grid rephasing matrices into the SRAM connected to the FPGA. The compression scheme described in Section 4.8.1 is used to allow for larger scan-grid sizes. The layout of the compressed matrices in memory is shown in Table 5.1. Each phase-step vector must be written in the correct numerical format. Individual vector components should be in a 32-bit two's complement format, with 30-bits specifying the fractional portion of each value. This conversion may be obtained by multiplying each value by 2^{30}, rounding, and then converting

to "int32" format. The scan-grid rephasing matrices only needed to be loaded once, when the co-processor system is initialized. Configuration of the scan-grid may be accomplished using the included MATLAB script: `hardSART_setScanGrid`.

Memory Offset	(Matrix Column, Scan-point number)	Vector Component
`0x000000`	(Column 1, Point 1)	Real
`0x000004`	(Column 1, Point 1)	Imaginary
`0x000008`	(Column 2, Point 1)	Real
`0x00000C`	(Column 2, Point 1)	Imaginary
⋮	⋮	⋮
`0x000078`	(Column 16, Point 1)	Real
`0x00007C`	(Column 16, Point 1)	Imaginary
`0x000080`	(Column 1, Point 2)	Real
`0x000084`	(Column 1, Point 2)	Imaginary
⋮	⋮	⋮
`0x08` $\times$ `X` + `0x80` $\times$ `Y`	(Column X+1, Point Y+1)	Real
`0x08` $\times$ `X` + `0x80` $\times$ `Y` + `0x04`	(Column X+1, Point Y+1)	Imag
⋮	⋮	⋮
`0x1FFFF8`	(Column 16, Point 16,384)	Real
`0x1FFFFC`	(Column 16, Point 16,384)	Imag

Table 5.1: Layout of compressed scan-grid rephasing matrices in memory. Memory locations are listed as byte offsets, relative to the SRAM base address of 0x200000.

5.2 Loading a Signal Matrix

New signal matrix data is loaded for each SART scan, every time the host wishes to perform a SART location estimate. Signal matrix data should also be loaded in a 32-bit two's complement format. The location of the binary point is irrelevant, as movement of its location will only result in a scaling of the singular value magnitudes. However, the signal matrix data should be scaled such that enough significant bits are maintained, and so that numerical overflow will not occur. For matrices with elements whose magnitudes are on the order of 1.0, the scale factor should be about 2^{25}. The layout of the signal matrix in memory is shown in Table 5.2.

Memory Offset	(Column, Row, Bank)
0x000000	(1, 1, 1)
0x000004	(2, 1, 1)
0x000008	(3, 1, 1)
⋮	⋮
0x0001FC	(128, 1, 1)
0x000200	(1, 2, 1)
0x000204	(2, 2, 1)
0x000208	(3, 2, 1)
⋮	⋮
0x001FFC	(128, 16, 1)
0x002000	(1, 1, 2)
0x002004	(2, 1, 2)
0x002008	(3, 1, 2)
⋮	⋮
0x003FFC	(128, 16, 2)

Table 5.2: Layout of signal matrix data in memory. Memory locations are listed as byte offsets, relative to the base addresses of the RAM blocks that hold the real (0x040000) and imaginary (0x080000) components of the matrix data. Two matrix banks are provided. One bank may be loaded while processing of data in the second bank occurs.

5.3 Triggering SART Calculations

Computations within the SART co-processor occur in bursts. During each burst, rephasing, QR decomposition, and bidiagonalization computations are conducted for up to 128 scan-grid points. Bursts may be triggered using the control and status register. The contents of the control and status register may be obtained by reading from memory location 0x000004. Bits 0 through 28 of the control and status register may be altered by writing to memory location 0x000004. Table 5.3 lists the function of all bits in the control and status register. The reset command should be issued to the SART co-processor before each burst is triggered. A reset command may be issued by writing any value to memory location 0x000000. Completion of the reset command may be confirmed by reading the contents of control and status register, and confirming that bit location 28 contains a 1.

Bit Place	Function	Direction
0	Scan start control	Write
1-12	Scan starting-point	Read/Write
13-24	Scan end-point	Read/Write
25	Result bank selection	Read/Write
26	Source bank selection	Read/Write
27	Not used	-
28	Reset complete indicator	Read
29	Not used	-
30	Scan complete indicator	Read
31	Not used	-

Table 5.3: Control and status register layout

In order to trigger a SART scan, control bits 0 through 26 should be written. Bit 0 should be set to 1. The assertion of this bit triggers the scanning process. Bits 1 through 12 should be used to indicate which point on the scan-grid should be scanned first. The SART co-processor will scan this point and all scan-grid points up to (but not including) the end point, which should be specified using bits 13 through 24. Because the result registers in each bidiagonalization module contain two banks (see Section 4.10.3), bit 25 should be used to indicate which bank should be used to store the results of the computations. Likewise, there are two signal matrix banks (see Section 4.8.2). Bit 26 should be used to indicate which bank the signal matrix should be read from.

For example, to trigger a scan that starts on scan point 0 and terminates on scan point 127, and that reads signal matrix data from bank 1 of the input buffer and writes computation results to bank 1 of the result buffer, the value `0x06100001` should be written to the control register. When the computations have been completed, bit 30 of the control register will be asserted.

5.4 Retrieving the Results

Results may be retrieved from the SART co-processor by reading from the result buffers in the bidiagonalization stage. Because of the round-robbin scheduling in the bidiagonalization stage, the results for consecutive scan-grid points will be spread across multiple buffers. For

example, buffer 1 will contain results for scan-grid points 1,2,9,10,17,18, etc., whereas buffer 2 will contain results for points 3,4,11,12,19,20, etc., and so on. For each scan-grid point, the result data consists of 31 values, which correspond to the 16 diagonal, and 15 super-diagonal elements of the bidiagonalized signal matrix. Due to the order in which these values are computed during the bidiagonalization process, the diagonal and super-diagonal elements are interleaved within memory. Figure 5.1 depicts how the elements of a bidiagonal matrix are mapped to the result buffer memory. The base addresses of the result buffers are listed in Table 5.4.

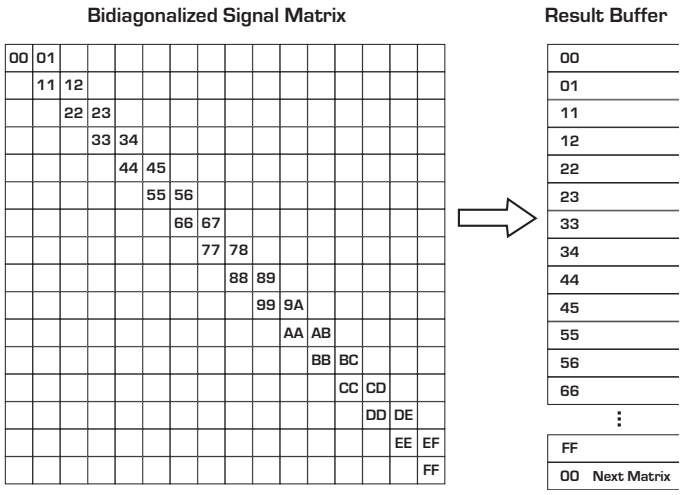

Figure 5.1: Mapping of bidiagonalized matrix into the result buffer

Buffer Number	Bank Number	Memory Offset
1	0	0x0C0000
2	0	0x100000
3	0	0x140000
4	0	0x180000
1	1	0x0C1000
2	1	0x101000
3	1	0x141000
4	1	0x181000

Table 5.4: Layout of result data in memory. Two result banks are provided. Results may be retrieved from one bank while new results are being generated and stored in the second bank.

5.5 Diagonalization

After a set of SART scan results have been retrieved, the final step in the SART algorithm, diagonalization, must be completed. This may be accomplished using the linear algebra library package, LAPACK. More specifically, the LAPACK function SBDSQR is an optimized function for computing the singular values of a single-precision bidiagonal matrix. A C program, which uses the SBDSQR function to process a batch of bidiagonal matrices was written expressly for this purpose. The program, `sbdsqr_mex`, was written using the "mex" protocol, which allows it to be called directly from MATLAB. The function may be called using the convention

```
s = sbdsqr_mex(n,D,U)
```

where n is the number of bidiagonal matrices to be processed, D contains the diagonal elements, and where U contains the super-diagonal elements of these matrices. The output of `sbdsqr_mex`, s, is an array containing the first singular value from each matrix. For example, if $n = 2$, then D should contain 32 elements and U should contain 30 elements. The first 16 elements of D and the first 15 elements of U belong to the first matrix and should be listed starting from its upper left corner. The remaining elements in D and U belong to the second matrix. D and U should be converted to single precision floating point numbers before being passed to the function. For the example case, the output array, s, will contain two single precision numbers. These are the first singular values of the two input matrices.

Chapter 6

Performance

This chapter contains information related to the performance of the SART co-processor system. There are two important metrics by which the performance of the system should be judged. Firstly, use of the co-processor system should not negatively impact the position estimates generated using the SART algorithm. Meeting this requirement implies that the system is capable of carrying out computations, in all processing stages, to a sufficient numerical accuracy. This would be guaranteed if the system employed the same single-precision floating point arithmetic that is currently used in the software implementation. However, because a fixed-point representation is used, and because the design of some stages involved speed/accuracy tradeoffs, the impact of these decisions must be assessed. In the first section of this chapter, the accuracies of arithmetic sub-modules within the system are tested individually, leading up to a test of metric computation accuracy.

The second important measure of performance relates to how quickly the SART co-processor system can complete a given set of computations. This rate can be compared to the rate of computation using a standard CPU, without a co-processor. The ratio of these rates is typically referred to as the "speedup". When examining the speedup, and how it might be improved without the used of additional hardware, the efficiency of the system becomes important. Anywhere a resource is left idle, e.g. while waiting for data from some other portion of the system, the efficiency and speedup-per-dollar is reduced. The second section in this chapter deals with speedup and efficiency performance.

6.1 Accuracy

This section presents the results of accuracy performance tests. The first two sub-sections deal with the accuracy of the arithmetic circuits from which the rest of the system is built. In the third sub-section, the accuracy to which the system computes the SART metric is discussed, and related back to the performance of the arithmetic modules.

6.1.1 Vector Processing Unit

In order to generate each component in the result of a vector operation, the VPU performs two multiplication operations, and then sums the results. This is accomplished using fixed point arithmetic. Because the multiplication of two fixed point numbers produces a result with twice as many bits, the result of the computation is rounded such that the output precision is equal to the input precision. From one perspective, the result of this operation may be compared to the result of the same operation conducted in floating point. Alternatively, the accuracy may be compared to the accuracy of a fixed-point computation conducted without output rounding. The first comparison is good for relating the accuracy of the co-processor to the accuracy of the software implementation. The second comparison is better for revealing rounding mistakes, which will appear in the form of errors greater than one half the value of a least-significant-bit in the rounded result. The test described here was used to insure that rounding was properly implemented. Examination of the effects of fixed vs. floating point arithmetic is left to Section 6.1.4.

In order to test the VPU, an automated VHDL testbench was constructed. The testbench may be run in simulation, because VHDL simulations are bit-for-bit and cycle-for-cycle accurate. Input data are generated in MATLAB, and saved to a file. The VHDL testbench program loads the test vector file, and applies its contents to the inputs of the VPU. The corresponding outputs are saved to a second file, which is then imported into MATLAB for analysis. The testbench exercises each of the VPU operations using input vectors with uniformly distributed magnitudes and phase angles. The results are compared to the ideal, un-rounded, fixed point results to insure proper rounding. Figures 6.1 through 6.6 show the results of the test. These histograms show the distribution of computation errors for 50,000 test vectors per operation. They show a uniform distribution in the interval [-0.5, 0.5], which

is the expected result for correctly implemented rounding. Correlation coefficients relating imaginary and real error components were on the order of 0.002, suggesting that they are independent, which they should be.

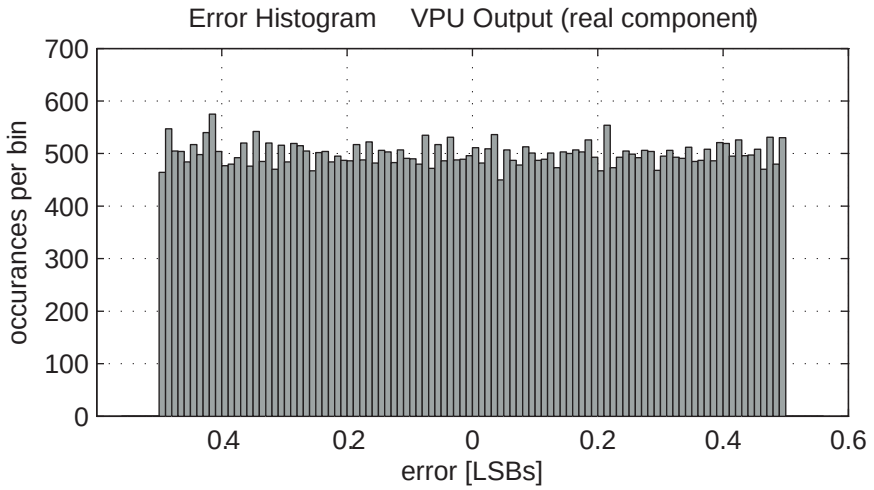

Figure 6.1: Histogram of error in real component of VPU output for "rotate" operation.

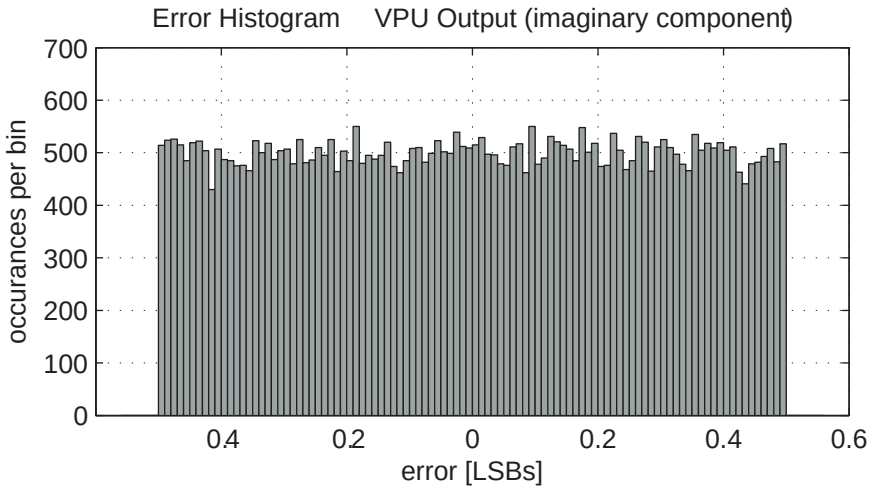

Figure 6.2: Histogram of error in imaginary component of VPU output for "rotate" operation.

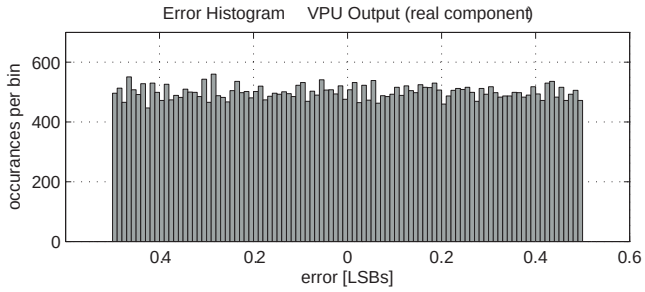

Figure 6.3: Histogram of error in real component of VPU output for "output" operation.

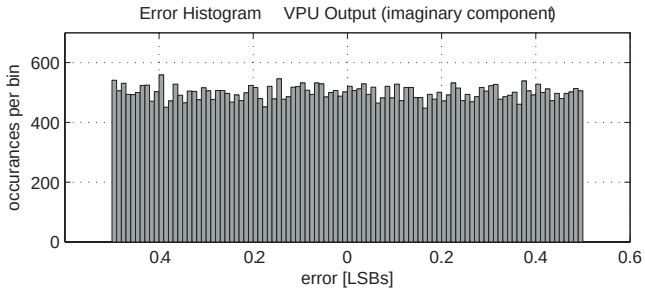

Figure 6.4: Histogram of error in imaginary component of VPU output for "output" operation.

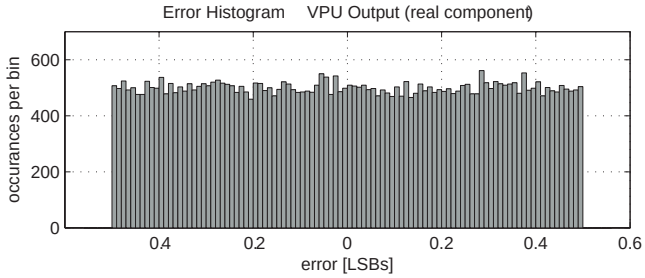

Figure 6.5: Histogram of error in real component of VPU output for "feedback" operation.

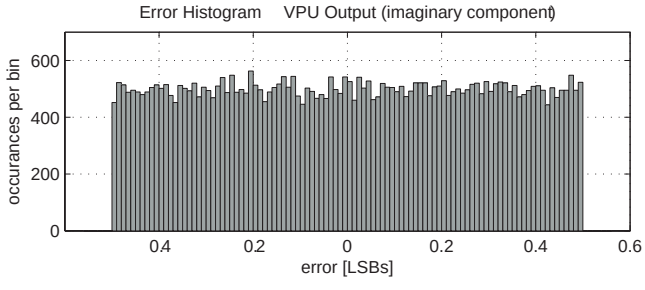

Figure 6.6: Histogram of error in imaginary component of VPU output for "feedback" operation.

6.1.2 CORDIC Module

The purpose of the CORDIC module is to generate the unit vectors that are used by the QR decomposition and bidiagonalization stages to perform Givens rotations. Errors in the angles of these vectors will lead to imperfect annihilation of matrix elements. Error in the vector magnitudes will produce errors in the magnitudes of rotated rows and columns. Both of these effects will lead to errors in the singular values computed using the co-processor system.

During the initial development of the SART co-processor system, a freely available CORDIC module from the Xilinx IP library was used. Eventually it became evident that this module consumed too many resources within the FPGA, leading to an overall design that didn't fit within a single FPGA. This would have been acceptable if all resources within the device were efficiently utilized, because additional devices could be added to form the required processing array size. However, the Xilinx design uses only CLBs, and does not use any DSP blocks. Because the SX55 FPGA contains many DSP blocks and relatively few CLBs, use of the Xilinx module would have meant leaving many DSP blocks idle. This would have been a very inefficient use of the available resources. For this reason, a new CORDIC module was designed. This module gains almost all of its functionality from DSP blocks, making it well suited to the SX55.

Each DSP block is pipelined so that maximum clock frequencies may be obtained. Unfortunately, the latency associated with this pipelining leads to a larger latency in the CORDIC module. Because the CORDIC algorithm is iterative, commencement of each iteration must be delayed until the result of the previous iteration exits the processing pipe. For this reason, the length of the pipeline was reduced as much as possible. In order to minimize the pipe latency, a trade-off was made between speed and accuracy. More specifically, the result of each shift operation is truncated rather than rounded off. Errors associated with truncation are minimized by choosing a truncation point that is below the LSB. This allows the accumulation of the values smaller than one LSB of output precision, which may eventually overflow into the LSB. This is better than truncating each shift operation to the output precision, but clearly worse than rounding. In order to obtain an empirical measure of the CORDIC module's accuracy a testbench was constructed to test its performance. Vectors of various magnitudes and phases were generated, and applied as input to CORDIC module.

The corresponding outputs were gathered and assessed for accuracy.

The first test of accuracy involved observing the magnitudes of the unit vectors generated given the testbench inputs. Ideally, these vectors should each have a magnitude of one. Figure 6.7 shows the errors in the unit vector magnitudes. The accuracy to which the CORDIC module measures the angles of input vectors was assessed indirectly. This was accomplished by multiplying the produced the unit vectors with their corresponding input vectors. The ideal result would be a set of vectors that lie along the real axis, and have magnitudes equal to their corresponding input vectors. The results of this test are shown in Figure 6.8 as a scatter plot, with the errors normalized with respect to one LSB. The third performance metric for the CORDIC module is how accurately it calculates the length of each input vector. Figure 6.9 shows a histogram of the errors in magnitude measurements relative one LSB. No attempt was made to analytically determine how any these errors might effect the overall capability of the co-processor system to compute the matrix singular values. Instead the accuracy performance of stages that employ the CORDIC module were measured empirically. These results, which are quite good, may be found in Section 6.1.4.

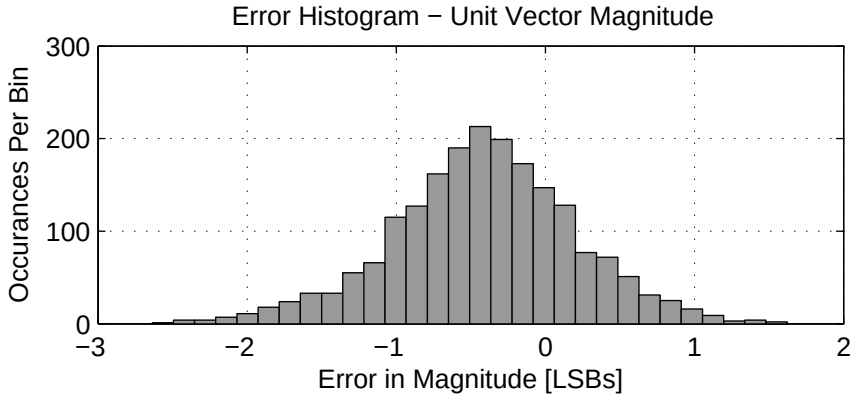

Figure 6.7: Histogram showing distribution of errors in unit vector magnitudes generated using the CORDIC module.

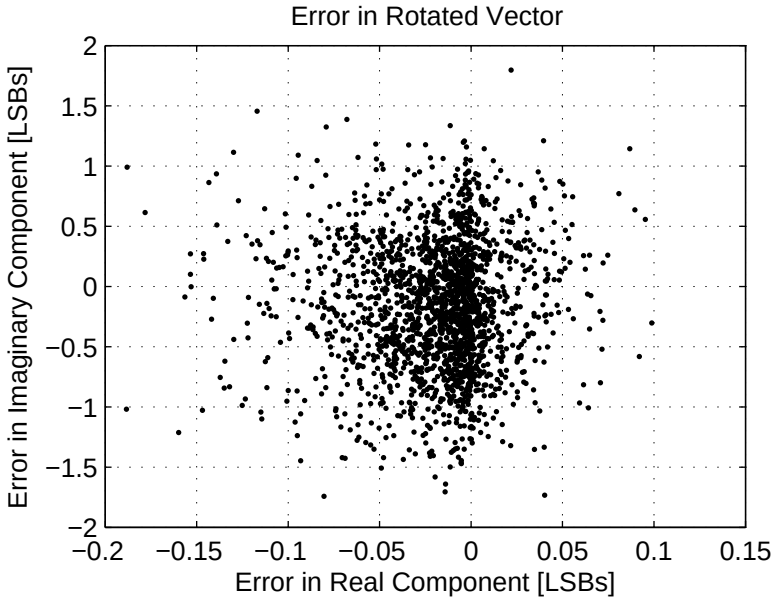

Figure 6.8: Scatter plots showing errors in vectors that have been rotated using unit vectors produced by the CORDIC module.

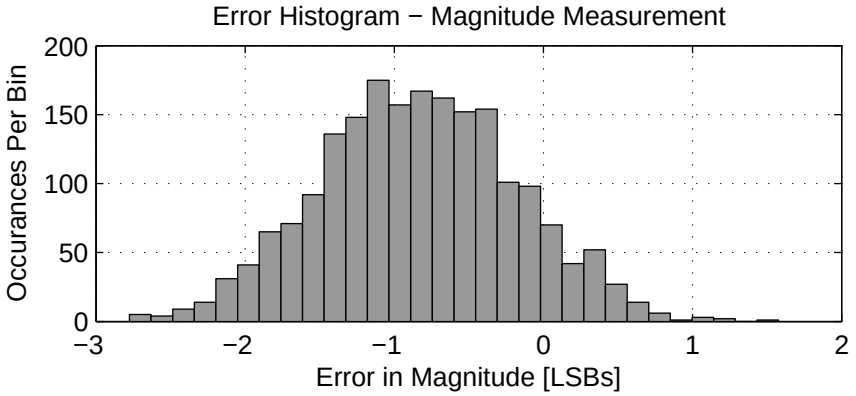

Figure 6.9: Histogram showing distribution of errors in magnitudes calculated using the CORDIC module.

6.1.3 Rephasing

The rephasing stage takes advantage of the regular spacing of the sub-carrier frequencies and the linear nature of the phase-frequency relationship to achieve excellent compression of SART rephasing matrices. The unit vectors used to rephase each sub-carrier are generated by multiplying a phase-step vector with the rephasing vector from the next lower subcarrier. Unfortunately, this iterative method leads to the accumulation of error in the rephasing vector due to the repeated multiplication of rounded numbers. Neglecting further rounding during the computation, a rough approximation of the worst case error in the magnitude of a unit-length rephasing vector, during the k^{th} iteration is

$$E_k = |1 - (1 + \epsilon)^k| \approx k\epsilon \tag{6.1}$$

where ϵ represents a very small error due to rounding. For a two component phase-step vector, the worst case error has a magnitude of 0.707 LSBs. Therefore, the expected worst case error in the k^{th} rephased sub-carrier is approximately 0.707k LSBs. For a signal with hundreds of carriers, this becomes fairly significant, 70 LSBs. The effect of this error on the final SART metric value was simulated using MATLAB. The phase step vectors were rounded to match the current precision of the rephasing stage. Erroneous rephasing matrices were generated from the rounded phase step vectors using the iterative decompression technique. These were then used to calculate a SART metric image with 4096 points. Other than the intentional rounding, the calculations were conducted in double precision floating point arithmetic. The results showed an approximately Gaussian distribution of error, at a RMS level of about -165 dB with respect to the Frobenius norm of the signal matrix. That is,

$$20log_{10}\left(\frac{1}{||\mathbf{S}||_F}\left[\frac{1}{n}\sum_{i=1}^{n}(\sigma_1 - \sigma_{1-TRUE})^2\right]^{\frac{1}{2}}\right) = -165 \tag{6.2}$$

where n is the number of points in the SART metric image, σ_1 is the erronious approximation of the actual first singular value, σ_{1-TRUE}, and $||\mathbf{S}||_F$ is the Frobenius norm of the signal matrix, $\mathbf{S}$. The next section will show that this is significantly lower then the errors in the SART metric solution, caused by inaccuracies in the CORDIC module output.

6.1.4 SART Metric Solution

In order to assess the accuracy of the overall SART metric computation, the co-processor system was used to calculate the singular values of various matrices. In the first two tests, sets of matrices with known singular values were constructed. In order to obtain statistical measure of accuracy, many matrices with identical singular values but randomized singular vectors were generated and analyzed with the SART co-processor. This was accomplished in MATLAB using the following method.

- Generate a matrix whose elements have uniformly distributed magnitude and phase
- Perform (double precision) singular value decomposition of the matrix to obtain randomized singular vectors $\mathbf{U}$, and $\mathbf{V}$, and corresponding singular values in $\mathbf{\Sigma}$.
- Alter $\mathbf{\Sigma}$ to contain the desired set of singular values.
- Calculate $\mathbf{U\Sigma V^H}$ to obtain a test matrix with the new set of singular values.

The first test involved matrices of rank 1, 2, 8, and 16, where all non-zero singular values were equal. The Frobenius norm of the signal matrix was swept to allow observation of the effect of input precision. The magnitude values were 2^{16}, 2^{20}, 2^{24}, 2^{28}, and 2^{29}. The number of test matrices was 1000 for each rank-magnitude combination. The results, which are displayed in the Tables 6.1 through 6.3, show that processing of the higher-rank matrices resulted in a larger error in the SART metric solution (the first singular value). This result makes sense, as the energy in the higher-rank matrices was spread across a greater number of singular values, thereby reducing their amplitude with respect to the minimum quantization level. The sweep in magnitude showed that this effect is reduced for matrices with large norms. For these matrices, quantization is less significant that internally generated errors . The source of these errors is the CORDIC module, which calculates rotation vectors to a limited precision. For singular value magnitudes of 2^{29} and greater, the error is very high. This is simply due to overflow of the CORDIC arithmetic.

	Rank			
Magnitude	1	2	8	16
2^{16}	1.0977	1.7949	5.4940	7.8270
2^{20}	1.3610	1.9653	5.9059	8.1779
2^{24}	1.6058	2.4559	6.1037	8.2802
2^{26}	4.1696	4.2096	6.8713	9.1379
2^{28}	15.905	12.600	11.424	12.651
2^{29}	25.7e6	4.65e6	17.703	18.320

Table 6.1: RMS error in SART metric (first singular value) for matrices of various rank, normalized with respect to one LSB. All non-zero singular values of test matrices were set equal. Their magnitude was varied to allow observation of the effect of input precision.

	Rank			
Magnitude	1	2	8	16
2^{16}	-95.52	-91.25	-81.53	-78.46
2^{20}	-117.7	-114.5	-105.0	-102.2
2^{24}	-140.4	-136.7	-128.8	-126.1
2^{26}	-144.1	-144.1	-139.8	-137.3
2^{28}	-144.5	-146.6	-147.4	-146.5
2^{29}	-26.38	-41.24	-149.6	-149.3

Table 6.2: Error in SART metric relative to its true value, expressed in decibels.

	Rank			
Magnitude	1	2	8	16
2^{16}	1.1024	1.0687	0.8079	0.8095
2^{20}	1.3639	1.1114	0.9447	0.6821
2^{24}	1.2115	1.1221	0.8913	0.7786
2^{26}	1.3612	1.0284	0.8498	0.8742
2^{28}	2.2856	1.5271	1.0209	0.8800
2^{29}	10.851	21.839	1.1430	1.0334

Table 6.3: Standard deviation of error in SART metric for test case 1, normalized with respect to one LSB.

For the second accuracy test, the singular values of the test matrices were not set to have equal magnitudes. Instead, the magnitudes of the singular values started at a maximum value and then fall off linearly to zero over the span of r singular values, where r is the matrix rank, such that the resulting Frobenious norm was 2^{16}, 2^{20}, 2^{24}, 2^{28}, or 2^{29}. The test results, which are shown in Tables 6.4 through 6.6, demonstrate that for high-rank matrices, the co-processor calculates the magnitude of the first singular value more accurately if the singular values have non-uniform sizes. This makes sense, because this implies that more energy is located in the first singular value, making quantization error less significant. Since this signal structure is not dissimilar to the structure of a real SART signal matrix, the results are quite encouraging. They suggest that the accuracy is about -145 dB with respect to the true value of the SART metric. This means that matrices whose columns exhibit a large dynamic range may be accurately processed using the SART co-processor system. This was confirmed in the third test, which is described below.

	Rank			
Magnitude	1	2	8	16
2^{16}	1.3984	1.1794	1.2197	1.4551
2^{20}	1.3186	1.2780	1.4646	1.3372
2^{24}	1.8986	1.5577	1.3502	1.4726
2^{25}	2.2051	2.0950	1.7630	1.3957
2^{26}	4.3184	3.7597	2.3955	2.1496
2^{27}	7.9616	6.5662	4.3646	3.3639
2^{28}	15.106	13.194	8.2097	6.8752
2^{29}	25.7e6	32.3e6	17.120	13.737

Table 6.4: RMS error in SART metric (first singular value) for matrices of various rank, normalized with respect to one LSB. Magnitudes of the non-zero singular values were equally spaced between 0 and listed magnitude.

	Rank			
Magnitude	1	2	8	16
2^{16}	-93.42	-94.90	-94.60	-93.07
2^{20}	-118.0	-118.3	-117.1	-117.9
2^{24}	-138.9	-140.6	-141.9	-141.1
2^{26}	-143.8	-145.0	-148.9	-149.9
2^{28}	-145.0	-146.2	-150.3	-151.8
2^{29}	-26.38	-24.39	-149.9	-151.8

Table 6.5: Error in SART metric relative to true value, expressed in decibels, for test case 2.

	Rank			
Magnitude	1	2	8	16
2^{16}	1.4028	1.1846	1.2255	1.4615
2^{20}	1.3248	1.2833	1.4370	1.3272
2^{24}	1.3131	1.2803	1.2181	1.3415
2^{26}	1.4910	1.2992	1.3616	1.3897
2^{28}	1.9288	1.8031	1.3657	1.6140
2^{29}	6.5097	17.9e6	2.0003	1.9269

Table 6.6: Standard deviation of error in SART metric for test case 2, normalized with respect to one LSB.

In order to observe the effect of quantization and round-off errors, the SART co-processor was tested using matrices whose columns differed greatly in magnitude. A worst case scenario, where one matrix column contains far more energy than any of the remaining columns, was assumed. This corresponds the case where one receive element in the PPL system receives a much stronger signal than the remaining antennas. This difference in signal strength, which for the sake of this discussion will be referred to as a power disparity, has the effect of reducing the relative magnitude of variations the SART metric. More specifically, in the case of a large power disparity, the fluctuations in the SART metric across different points on the scan-grid become very small with respect to its nominal value. This relationship was examined through simulation by computing the relative magnitude of variations in SART metrics, generated using a double precision floating point SVD in MATLAB, for signal matrices with various degrees of power disparity,

$$D = 20log_{10}\frac{||\overrightarrow{s_1}||_2}{||\overrightarrow{s_{k\neq 1}}||_2}, \tag{6.3}$$

where the test matrix, $\mathbf{S}$, has the form

$$\mathbf{S} = [\overrightarrow{s_1}\,\overrightarrow{s_2}\cdots\overrightarrow{s_n}] \tag{6.4}$$

and $\overrightarrow{s_k}$ is a column vector representing the signal received at the k^{th} antenna. That is, all columns of $\mathbf{S}$ have equal 2-norms apart from the first column, whose 2-norm differs according the power disparity, D. The results of the simulation are shown in Figure 6.10.

The curve shows that the relative magnitude of fluctuations in the metric decreases as power disparity increases. It suggests that power disparities above about 60 dB should be a problem

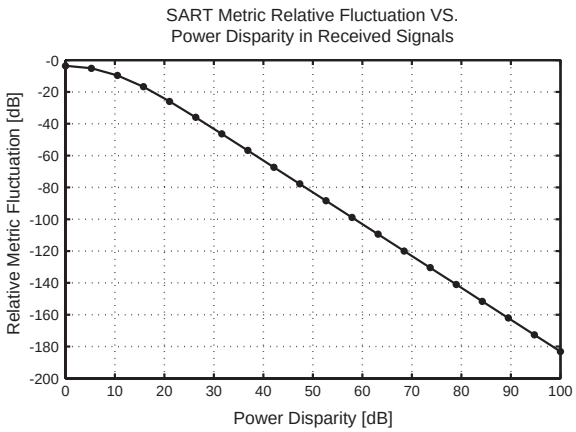

Figure 6.10: Simulation results showing that the relative magnitude of fluctuation in the SART metric decreases as the power disparity in the signal matrix increases.

given the -145 dB accuracy of the SART co-processor. This was confirmed by generating SART metric images from signal matrices with power disparities of 55, 65, 75, and 85 dB, using the co-processor. The results, which are displayed as contour plots in Figure 6.11, show that the performance does indeed degrade for power disparities greater than about 60 dB. This result was compared to a similar test, which employed single precision floating point computations instead of using the fixed point co-processor. The comparison shows that the co-processor preforms better than single precision floating point (see Figure 6.12); it can handle about 10 dB of additional power disparity. This is because the accuracy to which a numerical format can represent small fluctuations in a large value is limited by its mantissa length. Single precision floating point numbers have a mantissa of 24 bits. The co-processor system employs a minimum of precision of 31 bits, in the CORDIC module. Accumulation of truncation bias results in a performance that is comparable to about 25 or 26 bits of mantissa precision. Figure 6.13 shows the same test, performed using double precision floating point arithmetic. Because the double precision format includes 53 bits of mantissa precision, the performance is much better. The solution does not degrade until the signal matrix power disparity reaches about 135 dB.

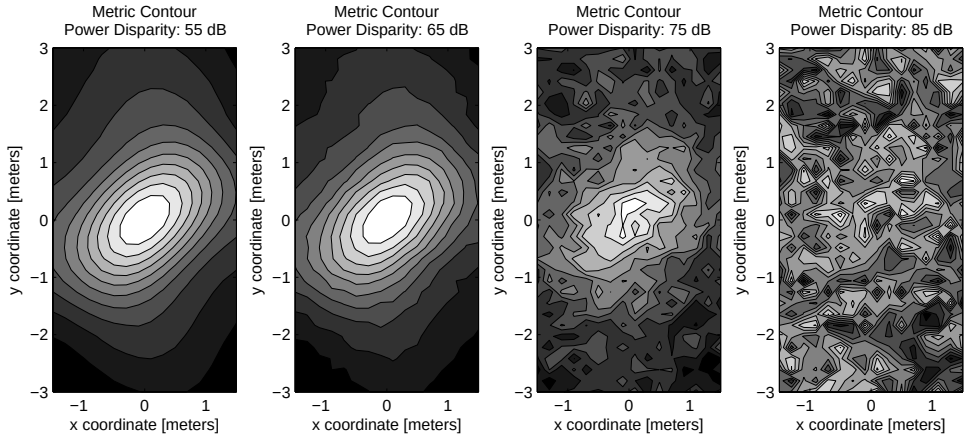

Figure 6.11: Contour plots showing the effect of power disparity between receive elements when using the SART co-processor. When one receive element (corresponding to one column in the signal matrix) receives a stronger signal than the remaining elements, fluctuations in the SART metric become small relative to its nominal value. In the extreme case, where the power disparity is a greater than about 60dB, quantization errors become significant.

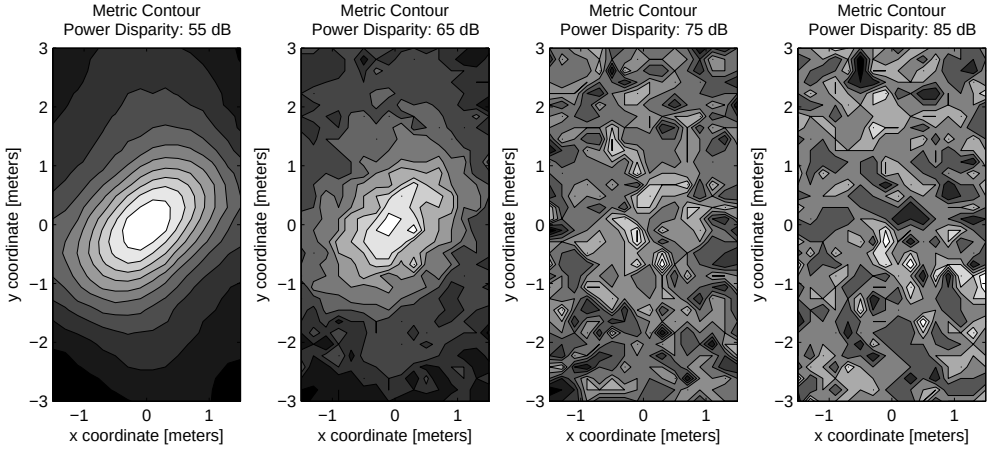

Figure 6.12: Contour plots showing the effect of power disparity between receive elements when single precision floating point arithmetic is employed. Because mantissa length is the limiting factor when representing small fluctuations in a large value, the results are not better than the accuracy of the fixed-point SART co-processor. In fact, they are worse. The SART co-processor is capable of tolerating about 10 dB of additional power disparity.

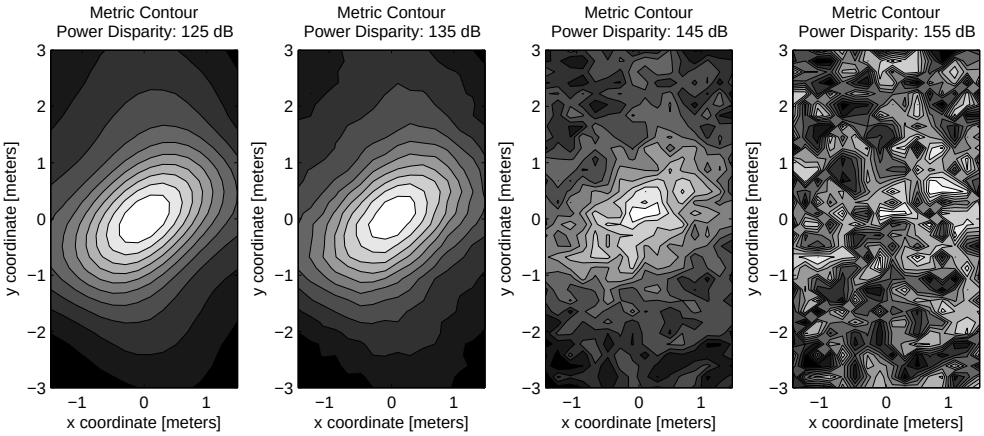

Figure 6.13: Contour plots showing the effect of power disparity between receive elements when *double* precision floating point arithmetic is employed. Because of the increased mantissa length associated with double precision, performance degradation does not begin until the power disparity reaches about 130 dB.

6.2 Speedup

In its current configuration the SART co-processor system provides a speed up of about 6 with respect to a Pentium 4 CPU running at 3 GHz. There are many factors that lead to this result. In the first sections of this chapter, the performance of the VPU and CORDIC sub-systems are revealed. This is followed by discussions concerning the performance that is achieved once these sub-system are combined to form the QR decomposition and bidiagonalization stages.

6.2.1 Vector Processing Unit

The VPU is a very efficient module. It is fully pipelined, and each DSP block within the VPU performs a necessary operation on every clock cycle when there is valid data in the processing pipe. The efficiency is therefore a perfect 1.0. The module is also capable of processing at high clock rates. The synthesis tools estimate the maximum clock frequency to be about 300 MHz.

6.2.2 CORDIC Module

The CORDIC module contains 8 DSP blocks. As shown in the CORDIC processing schedule (Figure 4.13), these blocks are used with an efficiency of only about 0.5. This is due to the fact that half of the DSP blocks are used only as input multiplexers, and are therefore active only for a few cycles at the beginning of a CORDIC computation. In a future version, this efficiency should be improved. According to estimates provided by the synthesis tools, the CORDIC module can be operated at a maximum clock frequency of 266 MHz.

6.2.3 QR Decomposition Stage

The efficiency of the QR decomposition stage is a function of the matrix dimensions. This is due to fact that the tasks of unit vector generation, in the CORDIC module, and unit

vector application (rotation), in the VPU, have differing computational requirements. More specifically, the CORDIC module requires a fixed number of clock cycles to perform its computation, whereas the duration of the VPU's computation depends on the width of the signal matrix. For the current implementation of the CORDIC algorithm, 150 cycles are required to perform the necessary calculations. Given the sequencing of calculations in the processing stage of the QR decomposition processing element, $3n$ cycles are required to process each new row of n elements. Therefore, perfect balance is not achieved unless the matrix width is 50. And even then, due to the 0.5 efficiency of the CORDIC module, the peak efficiency is only 0.75. For the current matrix width of 16, the efficiency (calculated as the ratio of active DSP blocks over the total number of DSP blocks in the design) is only 0.41. If the width of the signal matrix is not increased significantly in future implementations of the PPL system (i.e. if the number of antennas is not increased), then the balance between the CORDIC module and VPU should be improved. Figure 6.14 shows the approximate relationship between the efficiency of DSP block usage in the QR decomposition module and the width of the signal matrix.

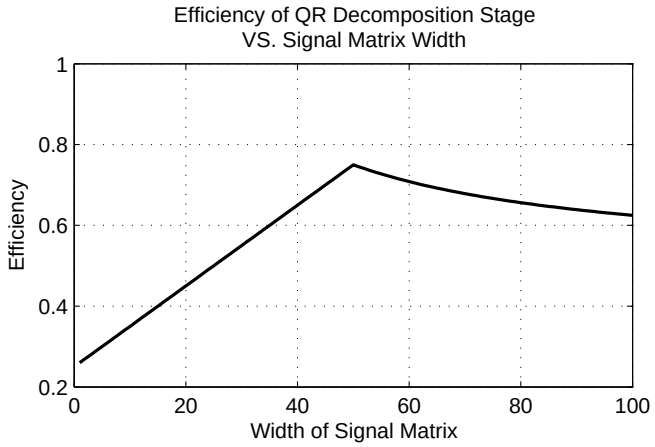

Figure 6.14: Curve showing the relationship between the efficiency of DSP block usage in the QR decomposition module and the width of the SART signal matrix.

One possible method would involve increased resource sharing, where each VPU processes more than a single row. This would require pairing each VPU with additional CORDIC modules, which would increase the amount of multiplexing required, thus increasing CLB usage. This may not be possible when targeting an SX series FPGA. If increased resource sharing was some how implemented on the SX55, and if each VPU processed two rows instead of one, then the efficiency could be increased to 0.54, and the overall QR decomposition speedup

could be doubled for matrices of width 16. This, however, would leave only enough DSP blocks for implementing bidiagonalization, and none would be left for the rephasing stage. This solution also neglects the inefficiency of the CORDIC module itself (see Section 6.2.2), which is why the overall efficiency is not dramatically improved. For these reasons it may prove beneficial to move to a different FPGA, with a different balance of resources, specifically one with more CLBs. Section 7.1 includes a discussion related to targeting different FPGAs.

6.2.4 Bidiagonalization Stage

Inefficiencies in the bidiagonalization stage arise for the same reason as in the QR decomposition stage. There is a lack of balance between the duration of the CORDIC computation and the duration of the operations performed by the VPU. Unlike in the QR decomposition stage, balance is not achieved for a larger matrix width. This occurs for the same reason that resource sharing was originally implemented in the QR decomposition stage: as elements in the matrix are annihilated, there are fewer elements to operate on. Therefore, the load on the VPU decreases as the matrix nears bidiagonal form, whereas the load on the CORDIC module remains fixed. This leads to an efficiency of about 0.27, which is quite low. There are a few possible ways around this.

Firstly, some sort of resource sharing could possibly be devised. For example, if multiple matrices were processed simultaneously, and if the processing was sequenced such that they produced complementarily sized loads on the VPU, then load balancing could be achieved. This, however, would be complicated, and would require more RAM for storing additional matrices. An alternate approach would be to serialize the computation. If the CORDIC operations and vector operations were implemented using the same set of DSP blocks, then they could be performed in sequence, rather than in parallel. This would make the relationship between their durations unimportant. This approach is certainly possible, because the CORDIC algorithm is simply a set of vector rotations. Furthermore, because this would mean that the CORDIC algorithm was being implemented with a full-precision multiplier (and not just a shifter), then improved algorithms, such as a high-radix CORDIC [29], could be implemented. The faster convergence of the high-radix algorithm may help offset the higher latency associated with full-precision multiplication.

Chapter 7

Conclusion

Singular Value Array Reconciliation Tomography (SART) is a method for locating a wide-band RF source which may be positioned within an indoor environment, where RF propagation characteristics make source localization very challenging. SART was developed in the Electrical and Computer Engineering department at the Worcester Polytechnic Institute (WPI) for the purpose of locating and tracking first response personnel such as firefighters. Unfortunately, the SART algorithm is very computationally intensive, due to the application of singular value decomposition. This thesis describes a co-processor system that has been designed to accelerate SART computations.

The co-processor system is based on field programmable gate array (FPGA) technology, which offers a low-cost alternative to customized integrated circuits, while still providing the high performance associated with a custom hardware implementation. Like custom IC's, FPGAs allow implementation of highly parallel systems. The selected Virtex-4 SX55 FPGA from Xilinx contains many optimized arithmetic circuits which be may operated simultaneously to obtain very high performance.

The SART co-processor system has been developed in the VHDL, and a prototype system has been implemented on a SX55 FPGA development platform from Alpha Data. The prototype system may be connected to the PCI bus of a standard PC. The system is easy to use, and may be accessed through a C program or MATLAB script. The co-processor is capable of computing the SART metric to an accuracy of about -145 dB with respect its true value,

which means that even relatively weak signals may make a meaningful contribution to the final SART solution. These results were shown to have slightly better accuracy than that obtained using single precision floating point arithmetic.

Compared to a Pentium 4 CPU running at 3 GHz, use of the co-processor system provides a speed-up of about 6 times for the current signal matrix size of 128-by-16. This performance could likely be doubled through future improvements in efficiency and clock rate. Using a single FPGA, a 10-by-10-by-10 meter volume may be scanned at 0.3 meter xy-resolution and 1.0 meter z-resolution at a rate of approximately 3 scans per second. Assuming an optimized scanning procedure that takes into account the previous location of a target, this figure roughly corresponds to tracking 3 first responders with a position update interval of 1 second. An arbitrary number of targets may be tracked if multiple FPGAs are used in parallel.

7.1 Future Work

The system described here represents a large amount of progress toward a final solution for SART acceleration. However, many improvements must be made before the system is ready for commercial implementation. Firstly, the system described here is a prototype, and has limited scalability. It is optimized for a matrix width of 16, and although the code of most processing stages is parameterized to allow flexibility in this area, some is not. Namely, the bidiagonalization stage must improved such that its memory structures automatically scale to accommodate the desired matrix dimensions. Furthermore, in order to process very large matrices, multiple FPGAs are needed. Although the co-processor was designed with this in mind, the actual chip-to-chip interface has not been designed.

Performance scalability may be achieved by operating multiple co-processors in parallel. Implementing this kind of system will not be a large technical challenge, but the manner in which data will be distributed within a multi-processor system must be considered. Similarly, the architecture presented here only includes implementations of some of the SART processing stages, assuming the remainder of the computations will be computed by the host. Despite the fact that the host performs only a small fraction of the total computations, if very high performance is necessary, then even this fraction may be more than the host can handle. In

this case, the final diagonalization stage should be moved into the co-processor. Alternately, each co-processor could be paired with a DSP processor, which may be better suited for performing the iterative task of diagonalization. Either of these improvements would give the co-processor system truly scalable processing power.

In addition to improvements in the area of scalability, it may be necessary to improve the accuracy of the SART co-processor. The current implementation of the co-processor produces metric calculations to an accuracy of about -145 dB with respect to their true value. This has been found to be sufficient for recently generated data sets, but may not necessarily be good enough. As the coverage area of the PPL system expands from a few dozen meters to a few hundred meters, the dynamic range of the received signal power across all receivers will likely increase. This means that some signals will be much weaker than others. In the case that one receive element is exposed to a very strong signal, while others receive much weaker signals, accuracy requirements will become even higher.

Better accuracy can be achieved in many ways. Firstly, various trade-offs were made, during the development of the current system, that were necessary to achieve efficient use of available resources, but which had a negative impact on overall accuracy. Most importantly, in the current implementation of the CORDIC algorithm, the result of each shift operation is truncated instead of being rounded. Because truncation bias accumulates over multiple operations, processing of larger matrices will result in lower accuracy. This module would make a good candidate for improvement. Another boost in accuracy may easily be obtained by improving the rephasing stage. As discussed in Section 4.8.1, the scan-grid phase reference matrices are stored in a compressed form within the co-processor system memory. Unfortunately, the iterative decompression technique magnifies small phase errors in the compressed data (see Section 6.1.3). This could be improved by increasing the precision of the decompression arithmetic, or by abandoning compression altogether and storing the phase reference matrices in raw form.

Improvements in the SART co-processor architecture may necessitate the use of a different FPGA. The SX55 is intended for signal processing applications, and compared to other devices in the Virtex-4 family, the ratio of arithmetic (DSP) blocks to the more generic configurable logic blocks is very high. This lead to design challenges. It was difficult to map certain algorithms to this type of coarse-grain architecture, specifically the CORDIC algorithm. In contrast, the largest device in the more general-purposed “LX” branch of the

Virtex-4 family has four times as many CLBs, but only one fifth as many DSP blocks. This device may suffer from the opposite problem, lack of arithmetic processing power. The FX series may achieve the best balance, with three times as many CLBs, and half as many DSP blocks as the SX55. The FX series devices are more expensive, but they have other enticing features. Each FX140 device contains 2 PowerPC processor cores, 4 gigabit ethernet MACs, and 24 high-speed IO interfaces. The processor cores may be used for sequential portions of the SART algorithm, such as the final diagonalization stage. The ethernet interfaces may be used to transfer data between the host and one or more co-processors. The high-speed IO may be used to interconnect multiple FPGAs when large matrices must be processed.

Figure 7.1 depicts a vision for the next generation SART processing system. The host system is connected to one or more co-processors through a robust and flexible ethernet network. This allows processing nodes to be located remotely using wired or wireless data links. For example, each emergency response vehicle might be equipped with enough processing power to track all of its occupants, yet receive its signal matrix data from a centralized host that applies pre-processing algorithms, such as synchronization [12]. Individual nodes will be optimized such that available resources are used efficiently, given the signal matrix size. This may involve a mix of devices. For example, FX series devices may be used to connect to the data distribution network using their ethernet MAC resources. By equipping their PowerPC cores with arithmetic accelerators, implemented using CLBs and DSP blocks, serialized portions of the SART algorithm may be executed rapidly. Remaining resources within the FX devices may be used to implement the rephasing and part of the QR decomposition processing array. For large matrices, the processing array may be extended using lower cost devices from the LX or SX series.

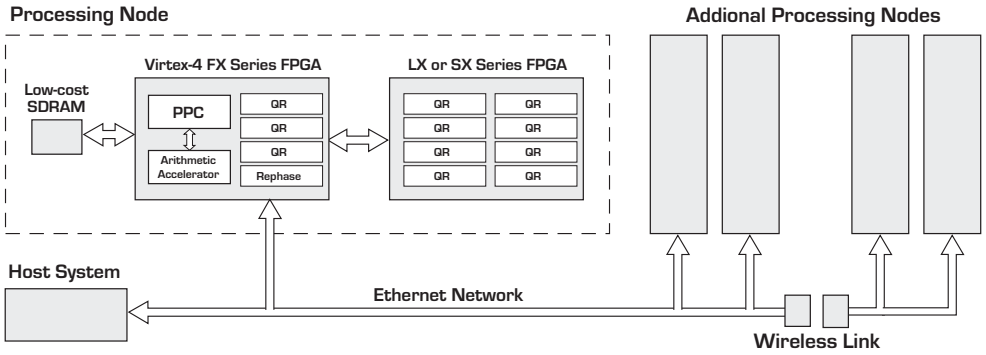

Figure 7.1: Next-generation SART processing system

Bibliography

[1] R. J. Fontana. Recent System Applications of Short-Pulse Ultra-Wideband (UWB) Technology. *IEEE Microwave Theory and Tech.*, 52(9), September 2004.

[2] A. Brown and Y. Lu. Indoor Navigation Test Results using an Integrated GPS/TOA/Inertial Navigation System. In *Proceedings of ION GNSS 2006*, Fort Worth, TX, September 2006.

[3] M. Rabinowitz and Jr. Spilker, J. J. A New Positioning System Using Television Synchronization Signals. White paper, Rosum Corporation, 301 North Whisman Road, Mountain View, California 94043, 2001.

[4] L. Ojeda and J. Borenstein. Non-GPS Navigation for Emergency Responders. In *International Joint Topical Meeting: Sharing Solutions for Emergencies and Hazardous Environments*, Salt Lake City, Utah, USA, February 2006.

[5] B. Alavi and K. Pahlavan. Modeling of the TOA-based Distance Measurement Error Using UWB Indoor Radio Measurements. *IEEE Communications Letters*, 10, April 2006.

[6] Zupt, LLC. Export Controls. http://www.zupt.com/export.htm.

[7] E. J. Canty. Six firefighters missing in blaze at vacant building. *Worcester Telegram & Gazette*, December 1999.

[8] D. Cyganski. WPI Precision Personnel Location (PPL) System. In *Institute of Navigation, 63rd Annual Meeting*, Cambridge, MA, April 2007.

[9] R. J. Duckworth; H. K. Parikh; W. R. Michalson. Radio Design and Performance Analysis of Multi Carrier-Ultrawideband (MC-UWB) Positioning System. In *Institute of Navigation, National Technical Meeting*, San Diego, CA, January 2005.

[10] K. Morris. COTS Supercomputing, DRC Harnesses FPGAs. *FPGA and Structured ASIC Journal*, July 2007.

[11] D. Cyganski. Patent application 60/934,880 - Singuar Value Array Reconciliation Tomography, 2007.

[12] V. T. Amendolare. Synchronization in an Indoor Precision Location System. Master's thesis, Worcester Polytechnic Institute, Worcester, MA, 2007.

[13] G. H. Gloub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, MD, 1996.

[14] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. ACM*, 5(4):339–342, 1958.

[15] W. Givens. Computation of plane unitary rotations transforming a general matrix to triangular form. *Journal of the Society for Industrial and Applied Mathematics*, 6(1), 1958.

[16] J. W. Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[17] R. Andraka. A survey of CORDIC algorithms for FPGA based computers. In *FPGA '98: Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, pages 191–200, New York, NY, USA, 1998. ACM Press.

[18] J. E. Volder. The cordic trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8:330–334, 1959.

[19] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[20] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM J. Matrix Anal. Appl.*, 16(1):79–92, 1995.

[21] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[22] M. Cornea; J. Harrison; P. T. P. Tang. Intel itanium floating-point architecture. White paper, Intel Corporation, 2003.

[23] D. Bursky. Designers stretched in ASIC, FPGA tug-of-war. *EE Times, embedded.com*, 2006.

[24] Xilinx Inc. Virtex-4 Data Sheet: DC and Switching Characteristics. Data sheet, Xilinx Inc., 2006.

[25] Xilinx Inc. Virtex-4 Users Guide. Users guide, Xilinx Inc., 2006.

[26] Xilinx Inc. XtremeDSP for Virtex-4 FPGAs User Guide. Users guide, Xilinx Inc., 2006.

[27] H. T. Kung. *Why systolic architectures?*, pages 300–309. IEEE Computer Society Press, Los Alamitos, CA, USA, 1986.

[28] C. M. Rader. Patent 4972361- folded linear systolic array, 1988.

[29] E. Antelo; T. Lang; J. D. Bruguera. Very-high radix CORDIC vectoring with scalings and selection by rounding. In Koren and Kornerup, editors, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pages 204–213, Los Alamitos, CA, 1999. IEEE Computer Society Press.